Valutazione di alcuni compilatori per High Performance Fortran

M. Aiello, R. Baraglia, R. Ferrini, D. Laforenza, R. Perego, M.C. Sechi
CNUCE - Istituto del Consiglio Nazionale delle Ricerche
Via S. Maria, 36 - I56100 Pisa (Italy)
Tel. +39-50-593111 - Fax +39-50-904052
e-mail: {R.Baraglia,R.Ferrini, D.Laforenza, R.Perego}@cnuce.cnr.it
URL: http://miles.cnuce.cnr.it

DRAFT

Progetto PQE2000
Unità Operativa CNUCE

Pisa, 30 Giugno 1997

Prefazione

In questo rapporto tecnico si descrivono le caratteristiche principali dei più diffusi compilatori per High Performance Fortran (HPF). Di alcuni di essi, in particolare `pghpf` del Portland Group, XL HPF della IBM, DEC Fortran 90 della Digital Equipment Corporation e ADAPTOR, sviluppato al GMD, viene riportata una valutazione relativa alla loro aderenza allo standard 1.1 e ad alcuni aspetti funzionali e prestazionali.

Ringraziamenti

Desideriamo ringraziare il CNUCE per le risorse tecniche forniteci, il Vienna Center for Parallel Computing (Dr.ssa Barbara Chapman) che ci ha permesso l'uso del sistema Meiko CS-2, e il Dr. Thomas Brandes del GMD per i preziosi suggerimenti ricevuti riguardo l'utilizzo di ADAPTOR.

Indice

Prefazione i

Ringraziamenti ii

1 I compilatori HPF 1

1.1 Il compilatore *pghpf* . . . 2
1.1.1 Sottoinsieme HPF implementato . . . 2
1.1.2 Estensioni al linguaggio HPF . . . 3
1.1.3 Debugging e profiling . . . 4
1.2 Il sistema di compilazione ADAPTOR . . . 5
1.2.1 Sottoinsieme HPF implementato . . . 5
1.2.2 Estensioni al linguaggio HPF . . . 6
1.2.3 Debugging e profiling . . . 6
1.2.4 Considerazioni . . . 6
1.3 Il compilatore XL HPF . . . 7
1.3.1 Sottoinsieme HPF implementato . . . 7
1.4 Il compilatore DEC Fortran 90 . . . 7
1.4.1 Sottoinsieme HPF implementato . . . 7

2 Confronto fra compilatori 9

2.1 Un primo approccio . . . 9
2.1.1 Confronto fra *pgphf* ed ADAPTOR . . . 10
2.1.2 Confronto fra i quattro compilatori . . . 15
2.2 PARKBENCH . . . 22
2.2.1 *pghpf* vs ADAPTOR . . . 23
2.2.2 Confronto fra tutti i compilatori . . . 25
2.3 Ulteriore confronto fra *pghpf* ed ADAPTOR . . . 28
2.3.1 Purdue Set . . . 28
2.3.2 Risultati delle prove . . . 29
2.4 Conclusioni . . . 31

A Listati dei programmi 33

A.1 Primi Passi . . . 33
A.2 PARKBENCH . . . 34
A.2.1 aa.hpf . . . 34
A.2.2 as2.hpf . . . 35
A.2.3 im.hpf . . . 36
A.2.4 ir.hpf . . . 38
A.2.5 it.hpf . . . 39
A.2.6 rd.hpf . . . 40

A.2.7 sh.hpf 41
A.2.8 st.hpf 42
A.2.9 tm.hpf 44
A.3 Purdue Set 45
A.3.1 Problem 01 45
A.3.2 Problem 02 46
A.3.3 Problem 03 47
A.3.4 Problem 04 48
A.3.5 Problem 05 49
A.3.6 Problem 06 51
A.3.7 Problem 07 53
A.3.8 Problem 08 54
A.3.9 Problem 09 56
A.3.10 Problem 11 58
A.3.11 Problem 12 59
A.3.12 Problem 13 60
A.3.13 Problem 14 61
A.3.14 Problem 15 64
A.4 Codice intermedio 67
A.4.1 Primi_Passi compilato con ADAPTOR 67
A.4.2 Primi_Passi compilato con *pghpf* 70

Elenco delle tabelle

1.1 Compilatori HPF disponibili (fonte HPFF) 2
1.2 Procedure di utilità fornite dal *pghpf* 4

2.1 Scalabilità di Primi_Passi con *pghpf* 20
2.2 Scalabilità di Primi_Passi con ADAPTOR 21
2.3 Scalabilità di Primi_Passi con XL HPF 21
2.4 Scalabilità di Primi_Passi con DEC Fortran 90 22
2.5 Parkbench con *pghpf* su Meiko CS-2 23
2.6 Parkbench con ADAPTOR su Meiko CS-2 24
2.7 Scalabilità dei PARKBENCH con DEC Fortran 90 26
2.8 Scalabilità dei PARKBENCH con *pghpf* 26
2.9 Scalabilità dei PARKBENCH con ADAPTOR 27
2.10 Scalabilità dei PARKBENCH con XL HPF 27

Elenco delle figure

2.1 Compilazione con *pghpf* di un costrutto FORALL 13
2.2 Compilazione con ADAPTOR di un costrutto FORALL 14
2.3 Diagramma delle dipendenze di due comandi FORALL (a) e di un costrutto INDEPENDENT FORALL (b) 15
2.4 Prima fase - *pghpf* su IBM SP2 16
2.5 Seconda fase - *pghpf* su IBM SP2 16
2.6 Terza fase - *pghpf* su IBM SP2 17
2.7 Quarta fase - *pghpf* su IBM SP2 17
2.8 Quinta fase - *pghpf* su IBM SP2 18
2.9 Sesta fase - *pghpf* su IBM SP2 18
2.10 Settima fase - *pghpf* su IBM SP2 19
2.11 Ottava fase - *pghpf* su IBM SP2 19
2.12 Speedup Settima fase - *pghpf* su IBM SP2 20
2.13 Scalabilità di Parkbench - *pghpf* 24
2.14 Scalabilità di Parkbench - ADAPTOR 25
2.15 Problem06 compilato con ADAPTOR e *pghpf* 29
2.16 Tempi di esecuzione del Problem07 30
2.17 Problem02 compilato con *pghpf* su Meiko CS-2 30
2.18 Problem08 compilato con *pghpf* 31
2.19 Scalabilità di Problem08 compilato con *pghpf* 31
2.20 Problem15 compilato con *pghpf* 32
2.21 Problem09 compilato con *pghpf* 32

Capitolo 1

I compilatori HPF

Al momento non sono molti i compilatori per HPF disponibili, e praticamente nessuno implementa tutte le caratteristiche definite nelle specifiche di HPF 1.1. Tutti i produttori comunque hanno rispettato i requisiti del *subset* HPF, tranne la Digital Equipment Corporation, e quasi tutti i prodotti implementano buona parte delle caratteristiche di *full* HPF, tranne alcune eccezioni, quali i puntatori e la distribuzione dinamica dei dati. In aggiunta, il *pghpf* ed ADAPTOR includono alcune caratteristiche che pur non facendo parte di HPF 1.1 sono, o sono state oggetto di discussione da parte di High Performance Fortran Forum. Molte di queste estensioni sono state introdotte nelle specifiche 2.0 di HPF.

Secondo HPFF [1] sono solo 4 i compilatori HPF attualmente disponibili (vedere tabella 1.1):

XL HPF Sviluppato da IBM, questo compilatore è disponibile solo per i sistemi IBM SP1 ed SP2 o su macchina virtuale MPI.

xHPF Sviluppato da Applied Parallel Research è disponibile per buona parte delle macchine parallele esistenti nonché per MPI e PVM.

pghpf Sviluppato dal Portland Group, è disponibile su moltissime architetture e supporta MPI e PVM.

Digital Fortran Sviluppato dalla Digital Equipment Corporation, è disponibile per la sola architettura Digital-Unix Alpha Systems.

A questa lista andrebbe aggiunto anche ADAPTOR, un sistema di compilazione sviluppato al GMD[2] dal gruppo diretto da Thomas Brandes, che non viene incluso in quanto è un traduttore codice–codice, ma per molti aspetti può essere considerato una ottima alternativa essendo di pubblico dominio.

Questo rapporto descrive le principali caratteristiche dei compilatori *pghpf*, `ADAPTOR`, `XL HPF` e `DEC FORTRAN 90`. Tutte le prove sono state effettuate sulle seguenti architetture:

- IBM SP2 ([CCL95], [SAH+], [IBMb], [IBMa]) con 8 processori e sistema operativo AIX versione 4.1, di proprietà dell'istituto CNUCE–CNR di Pisa, su cui è installato il compilatore ADAPTOR nella versione 4.0 ed XL HPF nella versione 1.1. Su tale

[1] La Home Page dello High Performance Fortran Forum è accessibile all'indirizzo `http://www.crpc.rice.edu/HPFF/home.html`, mentre una tabella riassuntiva dei compilatori disponibili con i link alle Home Page delle ditte produttrici si trova all'indirizzo `http://www.crpc.rice.edu/HPFF/hpfcompiler/index.html`

[2] German National Center for Computer Science, `http://www.gmd.de`. Un pagina dedicata interamente ad ADAPTOR ed al suo team di sviluppo è disponibile all'indirizzo `http://www.gmd.de/SCAI/lab/adaptor/adaptor_home.html`

Compilatore	Produttore	Target platforms
XL HPF	IBM	MPI, IBM-SP1, IBM-SP2
xHPF	Applied Parallel Research	MPI, PVM, IBM-SP1, IBM-SP2, Intel-iPSC860, Intel Paragon, Meiko-CS2, nCUBE-3, Cray-T3D, Hitachi-SR2001, Hitachi-SR2201, Fujitsu-VP300, Fujitsu-VP500
pghpf	Portland Group	MPI, PVM, IBM-SP2, Intel-Paragon, Intel-iPSC860, Meiko-CS2, Cray-T3D, SGI-PowerChallenge, nCUBE-3, Transtech-Paramid, CDAC-Param
Digital Fortran	Digital Equipment Co.	Digital-Unix Alpha Systems

Tabella 1.1: Compilatori HPF disponibili (fonte HPFF)

architettura è stato disponibile per un mese anche il compilatore *pghpf* , nella versione 2.0.

- MEIKO CS-2 ([Surc], [Surb], [Surd], [Sure], [Sura]) con 252 processori e sistema operativo SunOS versione 5.3, di proprietà del Vienna Center for Parallel Computing dell'Università di Vienna, su cui è installato il compilatore *pghpf* nella versione 2.1 ed ADAPTOR nella versione 4.0.
- DEC AlphaServer 2100A con 4 processori a memoria condivisa e sistema operativo Digital UNIX versione 4.0B, di proprietà dell'istituto CNUCE–CNR di Pisa, su cui è installato il compilatore DEC Fortran 90 nella versione 4.0.

Per i test eseguiti sia sulla SP2 che sul DEC, per ogni configurazione sono stati presi almeno otto tempi e di questi abbiamo considerato il minimo tempo di esecuzione. Tali tempi sono stati rilevati utilizzando la funzione intrinseca del Fortran 90 '`SYSTEM_CLOCK()`'.

1.1 Il compilatore *pghpf*

Il compilatore del Portland Group[3], giunto alla versione 2.1 ([Groc], [Grob]), implementa interamente il *subset* HPF, alcune caratteristiche di *full* HPF, alcune estensioni (PGI Language Extensions) e il supporto per procedure estrinseche scritte in FORTRAN 77.

1.1.1 Sottoinsieme HPF implementato

Nel manuale del compilatore *pghpf* si asserisce che il compilatore supporta tutte le caratteristiche del *full* HPF, tranne alcune eccezioni. Ci sembra eccessivo considerare semplici "eccezioni" l'attributo `PURE` e la clausola `NEW`, mentre si può accettare l'omissione di alcune

[3]The Portland Group, 9150 SW Pioneer Court, Suite H Wilsonville, Oregon 97070, http://www.pgroup.com/

caratteristiche Fortran 90 come i puntatori e le funzioni ricorsive, date le considerevoli difficoltà implementative di queste caratteristiche in un linguaggio parallelo. Vediamo di seguito le principali omissioni ai linguaggi Fortran 90 e HPF del compilatore *pghpf*:

- i puntatori del Fortran 90 non sono supportati e le parole chiave `POINTER` e `TARGET` non sono riconosciute;
- la ricorsione non è supportata;
- l'attributo `PURE` non è supportato;
- la direttiva `INDEPENDENT` ha un supporto limitato. In particolare, se applicata ad un costrutto `FORALL`, la direttiva `INDEPENDENT` non influenza le scelte del compilatore, così come la clausola `NEW`. Infine, la direttiva `INDEPENDENT` applicata ad un ciclo `DO` non ha effetto se il ciclo contiene un assegnamento ad array o un costrutto `FORALL` o anche un comando `ALLOCATE` o `DEALLOCATE`.

Per compensare queste carenze nel supporto alla direttiva `INDEPENDENT` il compilatore *pghpf* permette, a tempo di compilazione, di ottenere l'autoparallelizzazione di tutti i cicli e lo *inlining* per la parallelizzazione di cicli che contengono chiamate di procedure. La tecnica dell'*inlining* consiste nel sostituire, durante la compilazione, alla chiamata di procedura l'espansione del corpo della stessa opportunamente contestualizzata per garantire l'equivalenza semantica.

1.1.2 Estensioni al linguaggio HPF

Come già detto il compilatore *pghpf* introduce un insieme di estensioni chiamato PGI Language Extensions ed il supporto per procedure estrinseche FORTRAN 77.

PGI Language Extensions

Il PGI Language Extensions si riduce ad alcune estensioni e alle definizioni di strutture (PGI Structures and Records) che permettono la creazione di strutture dati formate da un insieme eterogeneo di variabili (Structures, Records) e strutture dati formate da più insiemi di variabili che alternativamente utilizzano la stessa area di memoria (Union e Map). Infine, l'insieme dei tipi disponibile viene esteso con la definizione di variabili puntatore, un tipo particolare di puntatore, e una nuova sintassi per le costanti ottali ed esadecimali. Per maggiori dettagli si veda [Grob] da pagina 237 a pagina 256.

Procedure estrinseche FORTRAN 77

L'estensione di maggior rilievo che il compilatore *pghpf* mette a disposizione dell'utente, è la possibilità di utilizzare procedure estrinseche FORTRAN 77. Con l'argomento `F77_LOCAL` si definisce una procedura estrinseca che rispecchia il modello di esecuzione delle procedure `HPF_LOCAL` ma, adottando la sintassi FORTRAN 77, permette di riutilizzare buona parte dei "serbatoi" di programmi e librerie già scritti in questo linguaggio e utili per velocizzare sezioni specializzate di codice, là dove il compilatore HPF può denunciare carenze in fase di ottimizzazione delle comunicazioni o a causa delle restrizioni del modello *data parallel*.

La procedura FORTRAN 77 chiamata può utilizzare il sistema di comunicazione sul quale si appoggia il compilatore o utilizzare una libreria di procedure di comunicazione fornite dal compilatore. Inoltre, sono fornite anche funzioni di interrogazione di sistema per ottenere

informazioni quali il numero di processori utilizzati per l'esecuzione e il numero del processore sul quale la procedura è in esecuzione. In tabella 1.2 sono elencate le procedure disponibili nell'attuale versione del compilatore. Per maggiori dettagli si veda [Groc] da pagina 73 a pagina 82.

Routines	Uso
`pghpf_nprocs()`	Numero di processori
`pghpf_myprocnum()`	Processore corrente
`pghpf_procnum_to_coord(...)`	Traduce dal numero di processore alle coordinate del processore nella griglia dei processori
`pghpf_coord_to_procnum(...)`	Dalle coordinate del processore nella griglia al numero di processore
`pghpf_csend(...)`	Send bloccante per l'invio di dati di tipo non–carattere
`pghpf_crecv(...)`	Receive bloccante per la ricezione di dati di tipo non–carattere
`pghpf_csendchar(...)`	Send bloccante per l'invio di dati di tipo carattere
`pghpf_crecvchar(...)`	Receive bloccante per la ricezione di dati di tipo carattere
`pghpf_tid(...)`	Traduce da numero di processore ad identificatore MPI o PVM del processore

Tabella 1.2: Procedure di utilità fornite dal *pghpf*

Compatibilità CM Fortran

Utilizzando una opzione del compilatore è possibile utilizzare istruzioni e procedure del linguaggio CM Fortran[4]. Sono supportate le intrinseche `DOTPRODUCT`, `DLBOUND`, `DUBOUND` e `DSHAPE`, il diverso metodo di definizione degli array (`ARRAY` invece dello standard Fortran 90 `DIMENSION`) e, per le intrinseche il cui nome sia uguale sia in CM Fortran che in Fortran 90, viene utilizzata la versione appropriata a seconda dei parametri di compilazione. Per una trattazione più approfondita si veda [Grob] da pagina 254 a pagina 256.

1.1.3 Debugging e profiling

Il compilatore *pghpf* è distribuito con lo strumento di profiling *pgprof* ([Groa]). Questo strumento permette l'analisi dei dati di traccia generati dall'esecuzione di programmi HPF compilati con l'opzione *-Mprof*. Il *pgprof* permette di ottenere informazioni sul tempo di esecuzione di una funzione o di una linea di programma, di conoscere quante volte viene invocata una procedura e anche di avere informazioni sulla distribuzione dei dati ai processori e sui *pattern* di comunicazione utilizzati.

[4] Fortran sviluppato dalla Thinking Machines Coorporation

Lo strumento *pgprof* è fornito con una interfaccia grafica (GUI) per permetterne un più semplice utilizzo ed una immediata comprensione dei risultati che vengono visualizzati mediante istogrammi oltre che con valori numerici.

Per quanto riguarda il debugging, non viene fornito nessuno strumento, e quindi bisogna utilizzare gli strumenti forniti dall'ambiente in cui si sta lavorando.

1.2 Il sistema di compilazione ADAPTOR

ADAPTOR (Automatic DAta Parallelism TranslatOR, [Bra96b], [Bra96d], [Bra96c]) viene presentato come un sistema di compilazione che traduce programmi *data parallel* scritti in HPF (e CM Fortran) in programmi FORTRAN 77 con chiamate alla libreria DALIB (Distributed Array LIBrary, [Bra96a]). Il codice risultante deve poi essere compilato, usando un qualunque compilatore FORTRAN 77, e in fase di linking, collegato con la libreria DALIB. Per quanto riguarda le primitive di comunicazione, ADAPTOR si basa sulla libreria DALIB, che può, quindi, essere riscritta utilizzando diversi sistemi. Con la versione 4.0 di ADAPTOR, la libreria DALIB viene fornita nelle seguenti versioni basate rispettivamente su:

1. PVM;
2. MPI;
3. comunicazioni tramite memoria condivisa;
4. CMMD, NX/2, EUI e ELAN per i sistemi CM-5, Intel Paragon, IBM SP e Meiko CS-2, rispettivamente.

1.2.1 Sottoinsieme HPF implementato

ADAPTOR nasce come uno strumento per la traduzione di programmi CM Fortran in programmi FORTRAN 77 con chiamate alla libreria DALIB. Con l'introduzione di HPF, ADAPTOR è stato adeguato a quest'ultimo, pur mantenendo la compatibilità col CM Fortran. In [Bra96b] si asserisce che ADAPTOR supporta, a parte alcune restrizioni, *full* HPF, con in più alcune caratteristiche proposte per HPF 2.0. Le principali caratteristiche non supportate da ADAPTOR rientrano nelle solite categorie di cui è nota la difficoltà di implementazione in ambiente parallelo. Non sono supportati:

1. i puntatori ad array distribuiti;
2. la distribuzione `CYCLIC(n)` con $n > 1$;
3. le funzioni di libreria HPF `xxx_PREFIX` e `xxx_SUFFIX`.

Queste limitazioni sono a prima vista minori rispetto a quelle presenti in altri compilatori. In realtà ADAPTOR non è in grado di tradurre in FORTRAN 77 programmi che contengano puntatori, tipi derivati, moduli e altre caratteristiche Fortran 90. Se si vogliono sfruttare tali caratteristiche del linguaggio di programmazione è necessario scegliere Fortran 90 come linguaggio di destinazione. Inoltre l'impossibilità di utilizzare distribuzioni `CYCLIC(n)` può degradare le prestazioni del codice generato.

Per quanto riguarda le procedure estrinseche, ADAPTOR supporta sia l'interfaccia `HPF_LOCAL` che `HPF_SERIAL`.

1.2.2 Estensioni al linguaggio HPF

Nella versione 4.0 ADAPTOR supporta le seguente estensioni approvate per HPF 2.0:

- distribuzioni irregolari;
- direttiva `REDUCTION`, per specificare le variabili di riduzione, cioè quelle variabili, in un ciclo `INDEPENDENT`, che saranno combinate alla fine del ciclo stesso;
- direttiva `ON HOME` per specificare su quale processore eseguire le iterazioni di un ciclo `INDEPENDENT`.

In aggiunta ADAPTOR introduce altre estensioni che sono ancora in fase di discussione per una loro inclusione nelle specifiche della prossima versione del linguaggio HPF:

- clausola `LOCAL_ACCESS` per i cicli `INDEPENDENT`, per specificare che i dati riferiti sono sicuramente locali;
- direttiva `SHARED` per allocare array distribuiti nella memoria condivisa (solo se l'architettura la supporta);
- direttiva `OVERLAP` per specificare l'area di overlap;
- direttiva `MAP` per mappare i processori astratti sui processori fisici.

1.2.3 Debugging e profiling

Pur non essendo un vero compilatore, ADAPTOR fornisce funzionalità di debugging, quali ad esempio la generazione di file che contengono informazioni riguardo le varie fasi della trasformazione. Per quanto riguarda il profiling ADAPTOR fornisce alcune funzioni per l'individuazione di sezioni di codice con un alto tasso di comunicazione e per la generazione dell'albero delle chiamate[5].

1.2.4 Considerazioni

In una pagina Web intitolata HPF Encyclopedia[6], alla voce compilatori HPF si legge che ADAPTOR "può girare su qualsiasi macchina da un PC compatibile ad una IBM SP2" e che "molti dicono che sia ragionevolmente buono considerando il suo costo".

A nostro avviso la definizione di semplice "traduttore" da HPF a FORTRAN 77 è restrittiva. Le principali caratteristiche che rendono questo prodotto di facile utilizzo e realmente portabile su ogni architettura sono: il fatto che è fornito in codice sorgente, la modularità con cui è stato progettato (libreria DALIB per implementare un'interfaccia con il sottosistema di comunicazione), e l'estrema disponibilità del team di sviluppo del GMD. In più il suo set di direttive ed istruzioni ci è sembrato il più completo oggi disponibile (specialmente se si utilizza Fortran 90 come linguaggio destinazione) e le estensioni di HPF 2.0 come `REDUCTION` e `MAP` realmente molto utili.

[5] Call Graph, è un albero in cui A è figlio di B se la procedura B "chiama" la procedura A

[6] `http://www.tc.cornell.edu/ bergmark/HPFstruff/HPFstuff.html`

1.3 Il compilatore XL HPF

La versione 1.0 del compilatore IBM XL HPF è stata annunciata il 14 Marzo 1996 con un documento[7], in cui si evidenziano le sue caratteristiche. Questo compilatore ([IBM96]) implementa tutte le caratteristiche del *subset* HPF.

Ad un primo approccio stupisce il fatto che non riconosca i file col suffisso `.hpf`, per cui è obbligatorio rinominare i file usando il suffisso `.f`. Utilizzando in fase di compilazione l'opzione `-qnohpf` il codice è trattato come un programma in Fortran 90, mentre con l'opzione `-qxlf77` si fornisce la compatibilità con le precedenti versioni del linguaggio XL Fortran.

Per quanto riguarda i protocolli di comunicazione viene sicuramente supportato MPI e il protocollo nativo dei sistemi IBM Scalable POWERparallel System (IBM SP1 e SP2). Non viene comunque fornita alcuna libreria di comunicazione e nel manuale non si parla della possibilità di utlizzare librerie standard, quali MPI e PVM, all'interno delle procedure estrinseche.

A corredo del compilatore XL HPF, IBM fornisce un debugger e uno strumento per la visualizzazione dell'esecuzione.

1.3.1 Sottoinsieme HPF implementato

Oltre al *subset* HPF, il compilatore della IBM include:

- le procedure `PURE`;
- il costrutto `FORALL`, completamente supportato;
- le direttive `SEQUENCE` e `NOSEQUENCE`;
- le procedure estrinseche `HPF_LOCAL` e `HPF_SERIAL`;
- alcune funzioni delle librerie `HPF_LIBRARY` e `HPF_LOCAL_LIBRARY`.

Inoltre tale compilatore permette l'uso della ricorsione e dei puntatori, ponendo però alcune limitazioni. I puntatori non possono essere il parametro formale di una procedura estrinseca `HPF_LOCAL` e non possono essere usati nell'ambiente di definizione di una procedura `EXTRINSIC(HPF)`. Le procedure dichiarate `EXTRINSIC(HPF_LOCAL)` non possono essere ricorsive.

1.4 Il compilatore DEC Fortran 90

Il compilatore DEC Fortran 90, come XL HPF, non riconosce i file col suffisso `.hpf`, ma accetta solo i suffissi `.f` o `.f90`. La scelta è condizionata dal fatto che il programma sia scritto nel formato fisso o libero.

1.4.1 Sottoinsieme HPF implementato

Tale compilatore supporta tutto il *subset* HPF (per maggiori dettagli vedere [Cor96]), tranne il costrutto `INDEPENDENT`. Oltre a questo include:

[7] `http://www.software.ibm.com/ad/fortran/xlhpf`

- la definizione di procedure PURE, con ulteriori restrizioni rispetto a quelle già definite da HPFF in [For94], come ad esempio il fatto che i nomi di variabili nell'ambiente di definizione della procedura PURE non possono essere esplicitamente distribuiti o allineati con l'utilizzo di una direttiva HPF;

- il costrutto FORALL ha un supporto limitato; al suo interno possono apparire solo degli assegnamenti, è permesso l'uso delle procedure dichiarate PURE o delle procedure intrinseche nella parte destra dell'assegnamento;

- le direttiva SEQUENCE e NOSEQUENCE;

- le procedure EXTRINSIC di tipo HPF_LOCAL ed HPF_SERIAL. Contrariamente alle restrizioni descritte in High Performance Fortran Language Specification [For94] il Digital Fortran 90 permette la chiamata ad una routine non–HPF, che sia FORTRAN 77 o C, dall'interno di una routine EXTRINSIC(HPF_LOCAL). La routine non–HPF deve essere dichiarata senza il prefisso EXTRINSIC. Esiste la limitazione che gli argomenti passati ad una procedura HPF_LOCAL non possono essere distribuiti CYCLIC(*n*).

Nelle specifiche del compilatore Digital Fortran 90, la direttiva INHERIT specifica che un parametro può essere allineato nello stesso modo di un parametro attuale, se tale parametro attuale è nominato in una direttiva ALIGN. Con tale compilatore è permesso ereditare da un parametro attuale solo un allineamento, non una distribuzione. La direttiva INHERIT non è quindi un sostituto della direttiva descrittiva DISTRIBUTE. Tale compilatore non accetta neanche la presenza di direttive trascrittive nell'interfaccia di una procedura (per chiarimenti sulle direttive prescrittive, descrittive e trascrittive vedere [SA96]).

Per l'utilizzo delle procedure EXTRINSIC non si fornisce alcuna libreria di comunicazione e nel manuale non è specificato come utilizzare altre librerie standard quali PVM ed MPI.

Capitolo 2

Confronto fra compilatori

I test effettuati si articolano in tre sezioni: un programma da noi realizzato, che ci ha permesso di confrontare la tecnica con cui il *pghpf* ed ADAPTOR generano il codice intermedio, oltre ovviamente al confronto fra la prestazioni degli eseguibili generati dai vari compilatori; un insieme di *kernel* della *suite* Parkbench; ed infine, per un ulteriore confronto fra *pghpf* ed ADAPTOR, abbiamo usato un gruppo di programmi di benchmark che contengono sezioni di codice appartenenti ad applicazioni reali.

2.1 Un primo approccio

Per iniziare lo studio dei compilatori presi in esame e valutare la scalabilità del codice generato, ed inoltre per cercare di comprendere i meccanismi che portano alla generazione del codice, si è pensato di implementare un programma HPF chiamato Primi_passi (il cui listato si trova in appendice A.1 a pagina 33). Il nome gli deriva dalla constatazione che si tratta del nostro primo programma HPF e dalla sua natura a fasi. Il programma è composto di tante fasi ognuna delle quali permette di comprendere il comportamento del compilatore in determinate situazioni. Da qui, per esempio, la scelta di inserire due fasi per il calcolo della stessa somma di vettori, la prima realizzata con un comando `FORALL`, la seconda con un ciclo `DO INDEPENDENT`. Questa prova ci ha permesso di scoprire che sia *pghpf* che ADAPTOR, trattano queste due fasi nello stesso modo, generando lo stesso codice intermedio FORTRAN 77.

Vediamo in dettaglio le otto fasi di cui si compone Primi_passi:

- Prima fase. Inizializzazione di tre vettori (a, b, c) in un unico ciclo `FORALL`.
- Seconda fase. c(i)=a(i)+b(i) in un ciclo `FORALL`.
- Terza fase. Come la seconda fase ma in un ciclo `INDEPENDENT DO`.
- Quarta fase. Come sopra ma con la sintassi Fortran 90:
  c(1:N)=a(1:N)+b(1:N).
- Quinta fase. c(i)=a(N-i+1)+b(i) in un ciclo `FORALL`. A differenza delle precedenti, in questa fase sono presenti molte comunicazioni.
- Sesta fase. Si calcola lo stesso risultato della quinta fase, ma con due cicli distinti: uno in cui d(i)=a(N-i+1) e uno in cui c(i)=d(i)+b(i).

- Settima e Ottava fase. Ridistribuzione dei vettori. Nella settima fase 4 vettori che avevano una distribuzione BLOCK vengono ridistribuiti CYCLIC, mentre nella ottava fase, uno solo di questi vettori viene ridistribuito BLOCK.

2.1.1 Confronto fra *pgphf* ed ADAPTOR

Analisi del codice intermedio

Anche se il compilatore *pghpf* è considerato un "vero" compilatore HPF, mentre ADAPTOR è descritto come un traduttore, in realtà entrambi generano un codice FORTRAN 77 con primitive di comunicazione che viene poi compilato con un compilatore FORTRAN 77 dell'architettura target e collegato con le librerie di comunicazione. Ad una prima occhiata si nota una sostanziale differenza tra i due prodotti. Il codice prodotto da ADAPTOR è più snello di quello generato da *pghpf* , il quale risulta molto più lungo e infarcito di variabili aggiunte dal compilatore. In compenso, il secondo è ben commentato e quindi non risulta problematico individuare le singole sezioni che compongono il programma. Di seguito sono riportate alcune sezioni di codice intermedio generate dai due compilatori. In appendice A.4 sono riportati i programmi completi.

Il codice seguente si riferisce alla compilazione di Primi_Passi con il compilatore *pghpf*.

```
 program primipassi
 integer n
 parameter (n=100000)
 common /pghpf_0/pghpf_0, /pghpf_1/pghpf_1, /pghpf_0c/pghpf_0c
+, /pghpf_lineno/pghpf_lineno, /pghpf_01/pghpf_01
 integer pghpf_0, pghpf_1, pghpf_lineno, pghpf_01(8)
 character*1 pghpf_0c
 common /prova$a$dyn/ a$dp,a$d$o,a$p,a$o,l$b5,u$b5
 common /prova$b$dyn/ b$dp,b$d$o,b$p,b$o,l$b6,u$b6
 common /prova$c$dyn/ c$dp,c$d$o,c$p,c$o,l$b7,u$b7
 common /prova$d$dyn/ d$dp,d$d$o,d$p,d$o,l$b8,u$b8
 .
 .
 .
 call pghpf_ptr_offset(c$d$o,c$dp,c$d,25)
 call pghpf_instance(c$dp,c$d(c$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(c$d$o,c$dp,c$d,25)
 call pghpf_template(d$dp,1,34048,sp$d(sp$dp),0,0,1,100000)
 call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
 call pghpf_instance(d$dp,d$d(d$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
 call pghpf_allobnds(a$d(a$d$o),l$b5,u$b5)
 call pgf90_alloc(u$b5-l$b5+1,27,pghpf_0,a$p,a$o,a)
 call pghpf_allobnds(b$d(b$d$o),l$b6,u$b6)
 call pgf90_alloc(u$b6-l$b6+1,27,pghpf_0,b$p,b$o,b)
 call pghpf_allobnds(c$d(c$d$o),l$b7,u$b7)
 call pgf90_alloc(u$b7-l$b7+1,27,pghpf_0,c$p,c$o,c)
 call pghpf_allobnds(d$d(d$d$o),l$b8,u$b8)
 call pgf90_alloc(u$b8-l$b8+1,27,pghpf_0,d$p,d$o,d)
 call pgf90io_src_info(10,'prova.hpf')
 z__io = pgf90io_ldw_init(6,0,0,0)
 z__io = pgf90io_ldw(14,1,0,'Numero di processori: ')
 z__io = pgf90io_ldw(25,1,0,hpf_np$)
 z__io = pgf90io_ldw_end()
```

```
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      continue
!     forall (i=1:100000)
      call pghpf_cyclic_loop(a$d(a$d$o),1,1,100000,1,c$l,c$u,c$s,c$lo9,
     +c$ls)
!     forall (i=i$l:i$u) a(i-c$lo-l$b5+a$o) = i
      c$lo = c$lo9
      do i$c9 = c$l, c$u, c$s
         call pghpf_block_loop(a$d(a$d$o),1,1,100000,1,i$c9,i$l,i$u)
         do i = i$l, i$u
            a(i-c$lo-l$b5+a$o) = i
         enddo
         c$lo = c$lo + c$ls
      enddo
!     a(i-c$lo-l$b5+a$o) = i
      call pghpf_cyclic_loop(b$d(b$d$o),1,1,100000,1,c$l1,c$u1,c$s1,
     +c$lo10,c$ls1)
```

I commenti (linee che iniziano con il simbolo "!") si riferiscono al codice HPF e permettono un facile orientamento nel codice anche se questo è poco leggibile a causa del massiccio uso di variabili aggiunte dal compilatore. Si noti inoltre la quantità di procedure che vengono invocate in fase di inizializzazione e le routine `pghpf_block_loop` e `pghpf_cyclic_loop` che traducono gli indici globali in indici locali utilizzati successivamente per riferire le partizioni di dati locali al processore su cui sono invocate.

Vediamo ora un esempio di codice intermedio generato da ADAPTOR.

```
      PROGRAM PRIMIPASSI
      INTEGER*4 N
      PARAMETER ( N = 100000 )
      INTEGER*4 SP_TOPID
      INTEGER*4 D_DSP
      REAL*4 D (1:2)
      .
      .
      .
      call dalib_init (4,4,4)
      call dalib_set_present (dalib_0)
      call dalib_start_subroutine ('PROVA',5)
      call dalib_top_create (SP_TOPID,1,1,dalib_nproc())
      call dalib_array_make_dsp (D_DSP,1,4)
      call dalib_distribute (D_DSP,SP_TOPID,1,0)
      call dalib_array_dynamic (D_DSP)
      call dalib_array_define (D_DSP,1,N)
      call dalib_array_allocate (D_DSP,D,D_ZERO,D_DIM1)
      call dalib_array_make_dsp (C_DSP,1,4)
      call dalib_distribute (C_DSP,SP_TOPID,1,0)
      call dalib_array_dynamic (C_DSP)
      call dalib_array_define (C_DSP,1,N)
      call dalib_array_allocate (C_DSP,C,C_ZERO,C_DIM1)
      call dalib_array_make_dsp (B_DSP,1,4)
      call dalib_distribute (B_DSP,SP_TOPID,1,0)
      call dalib_array_dynamic (B_DSP)
      call dalib_array_define (B_DSP,1,N)
      call dalib_array_allocate (B_DSP,B,B_ZERO,B_DIM1)
      call dalib_array_make_dsp (A_DSP,1,4)
```

```
call dalib_distribute (A_DSP,SP_TOPID,1,0)
call dalib_array_dynamic (A_DSP)
call dalib_array_define (A_DSP,1,N)
call dalib_array_allocate (A_DSP,A,A_ZERO,A_DIM1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Numero di processori: ',dalib_nproc()
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_lslice (A_DSP,1,1,N,A_START1,A_STOP1)
DO I=A_START1,A_STOP1
   A(A_ZERO+I) = REAL(I)
END DO
call dalib_array_lslice (B_DSP,1,1,N,B_START1,B_STOP1)
DO I=B_START1,B_STOP1
   B(B_ZERO+I) = REAL(I)
END DO
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = REAL(I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
```

ADAPTOR, a differenza di *pghpf* , non inserisce commenti, ma il codice intermedio da esso generato risulta ugualmente molto leggibile data la sua più semplice strutturazione. ADAPTOR infarcisce il codice con chiamate alla libreria DALIB e, come il *pghpf* delega ad un unico processore lo Input/Output.

Dallo studio del codice intermedio generato dai due compilatori se ne deduce che entrambi trattano i cicli `FORALL` e `INDEPENDENT DO` allo stesso modo e altrettanto fanno per gli assegnamenti con la sintassi Fortran 90. Inoltre entrambi non utilizzano completamente le potenzialità del costrutto `INDEPENDENT FORALL`, in quanto spezzano i costrutti `FORALL`. Per esempio, un costrutto `FORALL` con tre assegnamenti viene eseguito come tre comandi `FORALL`, ognuno con un assegnamento, introducendo delle sincronizzazioni non necessarie. In figura 2.1 è mostrato un esempio di codice intermedio, risultante dalla compilazione con *pghpf* di un costrutto `FORALL` con 3 assegnamenti al suo interno; in figura 2.2 è mostrato il corrispondente codice compilato con ADAPTOR. Come si può vedere il costrutto viene diviso in 3 cicli `DO` successivi.

In figura 2.3a è mostrato il diagramma delle dipendenze relativo a due comandi `FORALL` contenenti due assegnamenti ad array:

```
FORALL (i=1:N)
     a(i)=...
END FORALL
FORALL (i=1:N)
     b(i)=...
END FORALL
```

Come si può vedere questo diagramma contiene un punto di sincronizzazione che invece viene eliminato nel diagramma di figura 2.3b che si riferisce ad un costrutto `INDEPENDENT FORALL` al cui interno sono inseriti gli stessi assegnamenti:

```
Codice HPF
FORALL(i=1:N)
     a(i)=i
     b(i)=i
     c(i)=i
END FORALL

Codice Intermedio generato dal compilatore pghpf
...
!  forall (i=i$l:i$u) a(i-c$lo-l$b5+a$o) = i
     do i = i$l, i$u
          a(i-c$lo-l$b5+a$o) = i
     enddo
...
!  forall (i=i$l1:i$u1) b(i-c$lo1-l$b6+b$o) = i
     do i = i$l1, i$u1
          b(i-c$lo1-l$b6+b$o) = i
     enddo
...
!  forall (i=i$l2:i$u2) c(i-c$lo2-l$b7+c$o) = i
     do i = i$l2, i$u2
          c(i-c$lo2-l$b7+c$o) = i
     enddo
...
```

Figura 2.1: Esempio di compilazione con *pghpf* di un costrutto FORALL

```
!HPF$ INDEPENDENT
      FORALL (i=1:N)
           a(i)=...
           b(i)=...
      END FORALL
```

Nel primo caso tutti i processi vengono sincronizzati prima di poter iniziare l'elaborazione sulla matrice b, mentre nel secondo caso questo non avviene.

Risultati delle prove

Su di un'architettura IBM SP2 con 8 processori utilizzando il compilatore *pghpf* si sono ottenuti gli andamenti riportati nelle figure dalla 2.4 alla 2.11 per quanto riguarda il tempo di completamento delle varie fasi. Tutte le prove sono state effettuate con macchina in normali condizioni di carico, cioè con più processi contemporaneamente in esecuzione su ogni nodo. È stato quindi necessario effettuare più test per meglio evidenziare l'andamento generale della scalabilità ed isolare risultati determinati da uno sbilanciamento generale del carico sulla macchina. Nelle figure le differenti esecuzioni, effettuate al variare del carico sulla macchina, sono evidenziate da differenti simboli sul grafico.

Dai grafici è evidente che la scalabilità delle prime quattro fasi è buona (rispettivamente da figura 2.4 a figura 2.7). Questo perché le operazione eseguite non necessitano di comunicazioni. La scalabilità ottenuta è lineare o quasi in tutte e quattro le fasi. Questi test dimostrano che il compilatore, se pur "infarcisce" il codice con molte chiamate a procedure per l'inizializzazione e per il calcolo degli elementi locali, genera un codice con ottime prestazioni.

```
Codice HPF
FORALL(i=1:N)
     a(i)=i
     b(i)=i
     c(i)=i
END FORALL

Codice Intermedio generato dal compilatore ADAPTOR
...
DO I=A_START1,A_STOP1
     A(A_ZERO+I) = REAL(I)
END DO
...
DO I=B_START1,B_STOP1
     B(B_ZERO+I) = REAL(I)
END DO
...
DO I=C_START1,C_STOP1
     C(C_ZERO+I) = REAL(I)
END DO
...
```

Figura 2.2: Esempio di compilazione con ADAPTOR di un costrutto `FORALL`

Le fasi cinque e sei (rispettivamente figura 2.8 e figura 2.9) invece, necessitano per la loro elaborazione di comunicazioni. Il *pattern* di comunicazioni utilizzato è però alquanto semplice e ci si aspetterebbe ugualmente un miglioramento del tempo di completamento all'aumentare del numero di processori usati. Questo purtroppo non si verifica, i tempi misurati hanno un andamento caotico e non si ha scalabilità in quanto all'aumentare del numero dei processori utilizzati il tempo di completamento non diminuisce. La sesta fase ha riportato in molte prove una scalabilità leggermente migliore della quinta, sebbene siano semanticamente equivalenti. Questo potrebbe indicare un riutilizzo dei *pattern* di comunicazione. Infatti, per il calcolo delle fasi cinque e sei sono necessarie le stesse comunicazioni, e quindi il supporto a *run-time* del compilatore può evitare di ripetere i calcoli per la determinazione delle comunicazioni.

Infine le fasi settima e ottava, che contengono solo ridistribuzioni di matrici (solo comunicazioni), presentano le migliori sorprese. Infatti, come mostrato in figura 2.10 e 2.11, il tempo di completamento per l'esecuzione su 6 processori risulta 3 volte minore dell'esecuzione su 2 processori anche se naturalmente superiore all'esecuzione su un solo processore. L'andamento della scalabilità della settima fase si esprime meglio riferendosi alla scalabilità rispetto all'esecuzione su due processori (fig. 2.12). La scalabilità è buona, denotando una corretta implementazione dei pattern di comunicazione, così da permettere che comunicazioni tra coppie diverse di processori vengano eseguite in parallelo, ed evidenziando inoltre l'uso di un buon algoritmo di *scheduling*.

Sempre sul IBM SP2, ADAPTOR ha dato dei risultati molto simili, con la differenza che nelle fasi di ridistribuzione (settima e ottava) non si ha un miglioramento del tempo di completamento all'aumentare dei processori come con il *pghpf*. Ciò può indicare una realizzazione della fase di comunicazione strutturata derivante da una ridistribuzione non completamente ottimizzata. In compenso ADAPTOR è risultato senza dubbio più veloce, con tempi anche 4/5 volte inferiori rispetto a quelli del *pghpf*. Bisogna comunque dire che per le compilazioni fatte con *pghpf* non sono state abilitate le ottimizzazioni. Purtroppo non è stato possibile ripetere i test con le ottimizzazioni abilitate in quanto il compilatore è stato

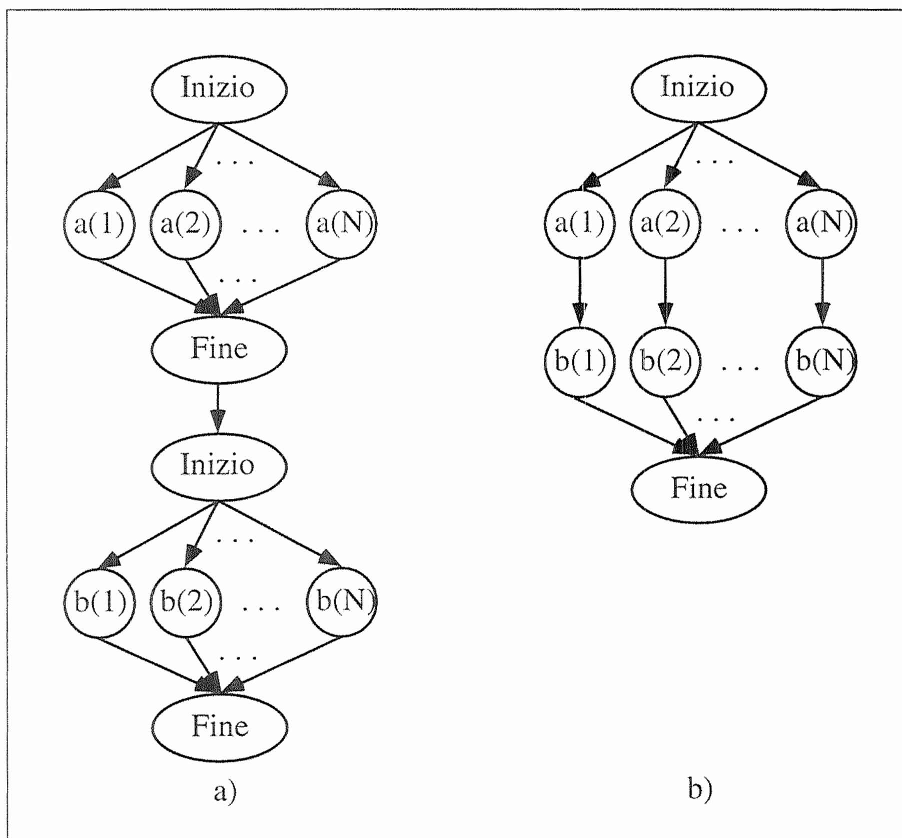

Figura 2.3: Diagramma delle dipendenze di due comandi `FORALL` (a) e di un costrutto `INDEPENDENT FORALL` (b)

disponibile solo in prova per un mese.

Sulla Meiko CS-2 è stato possibile effettuare altri test per valutare il guadagno che si ottiene nell'esecuzione di programmi compilati con *pghpf* abilitando le ottimizzazioni. Da notare che il Portland Group non supporta questa architettura e infatti non viene utilizzata la libreria di comunicazione Elan della CS-2 e il codice deve essere compilato utilizzando MPI. Se non si specifica l'opzione *-Mmpi* in fase di compilazione viene generato un codice con scalabilità nulla e tempi di esecuzione molto alti. I risultati ottenuti non mostrano grandi differenze tra l'esecuzione con o senza ottimizzazioni. Questo perché le varie fasi di calcolo si compongono di poche istruzioni e quindi non sono possibili ottimizzazioni. Da notare comunque come il compilatore eviti di ripetere più volte lo stesso calcolo. Infatti il programma Primi_Passi contiene tre fasi di calcolo della somma degli stessi vettori (seconda, terza e quarta fase). Utilizzando le ottimizzazioni in fase di compilazione, si ottengono tempi di esecuzione nulli per due di queste tre fasi.

2.1.2 Confronto fra i quattro compilatori

L'unico possibile confronto che si può effettuare fra differenti compilatori su differenti architetture è il confronto della scalabilità fra le varie fasi. Infatti per questo confronto abbiamo utilizzato i risultati dell'esecuzione del programma compilato con *pghpf* e ADAPTOR per la Meiko CS–2, con XL HPF per la IBM SP2 e con Digital Fortran 90 per il DEC Alpha System.

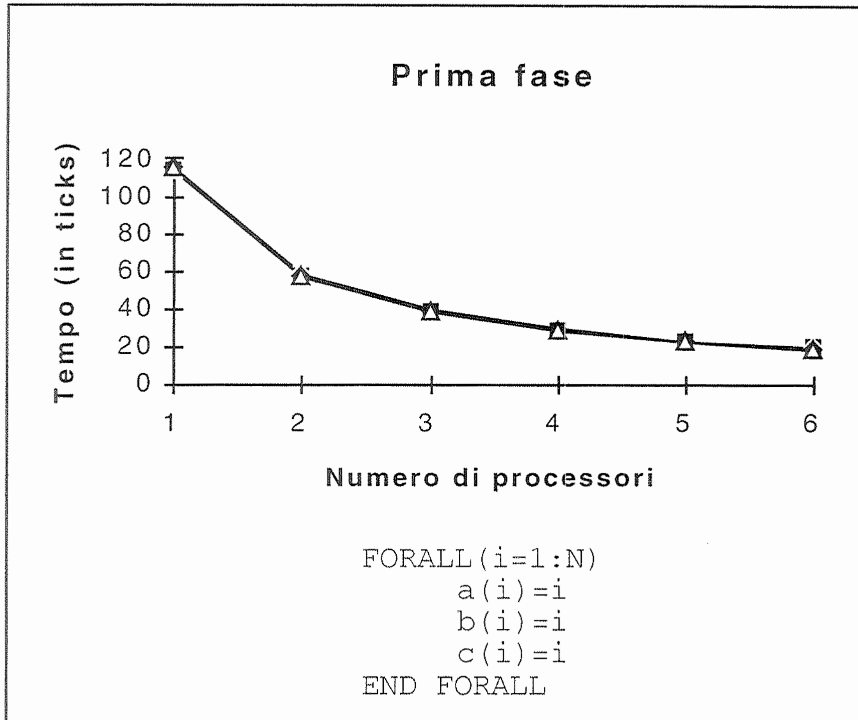

Figura 2.4: Prima fase - *pghpf* su IBM SP2

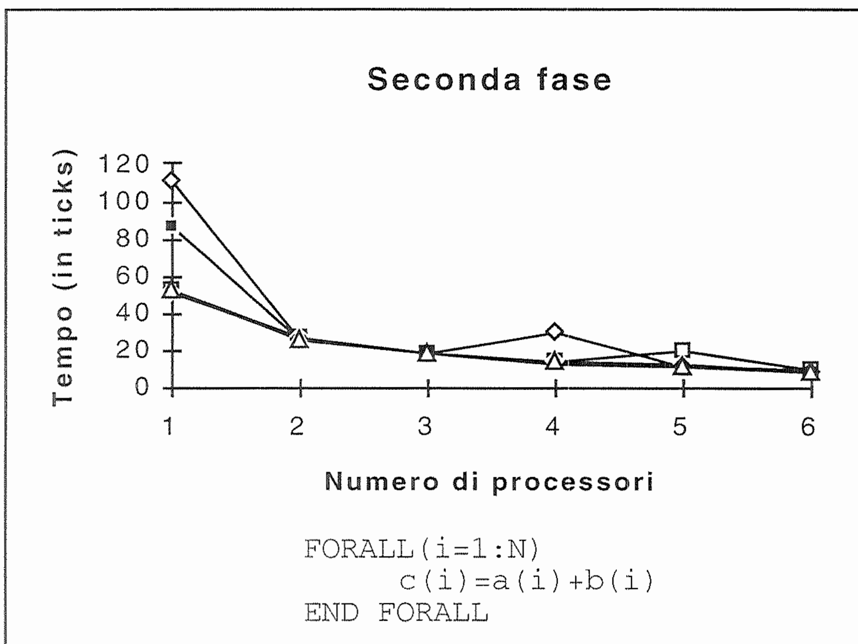

Figura 2.5: Seconda fase - *pghpf* su IBM SP2

Per un più approfondito esame fra i quattro compilatori sono state aggiunte altre due fasi, che sono state utilizzate per esplorare il comportamento dei compilatori di fronte all'indirizzamento indiretto:

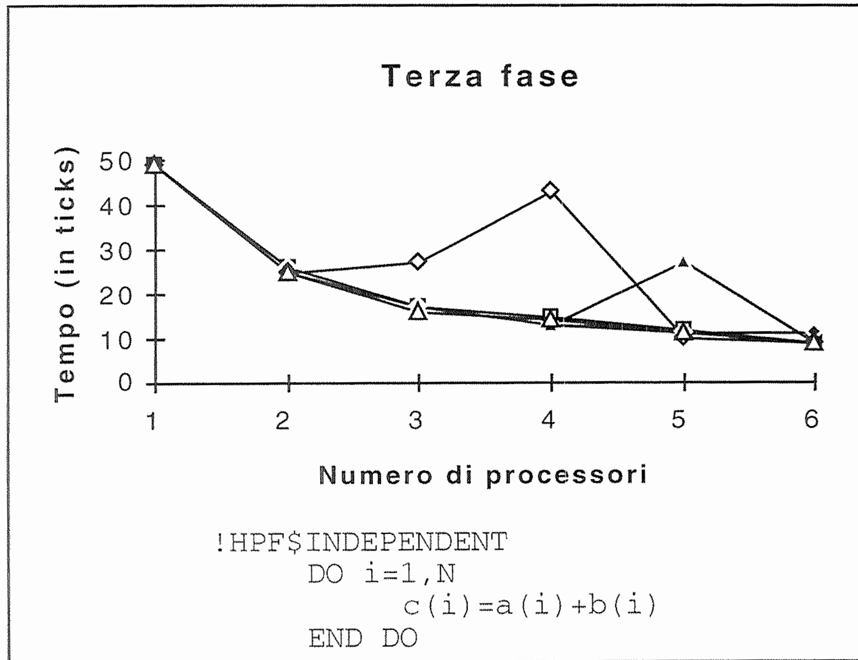

Figura 2.6: Terza fase - *pghpf* su IBM SP2

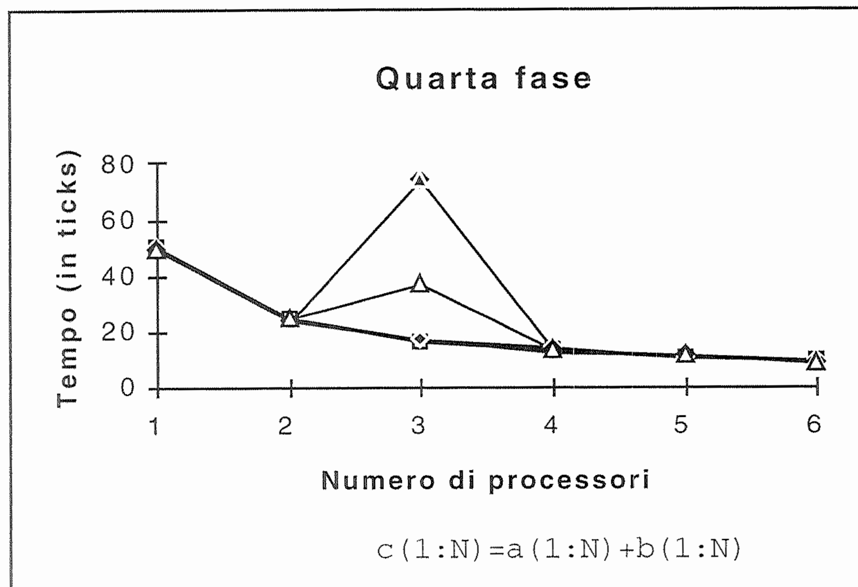

Figura 2.7: Quarta fase - *pghpf* su IBM SP2

- Nona fase. Indirizzamento indiretto: a(e(1:N))=c(1:N), in cui il vettore e è distribuito a blocchi fra i processori.
- Decima fase. Indirizzamento indiretto: a(f(1:N))=c(1:N), in cui non è stata definita alcuna distribuzione per il vettore f. Nel caso non sia definita alcuna distribuzione per un vettore i compilatori usati lo replicano su tutti i processori.

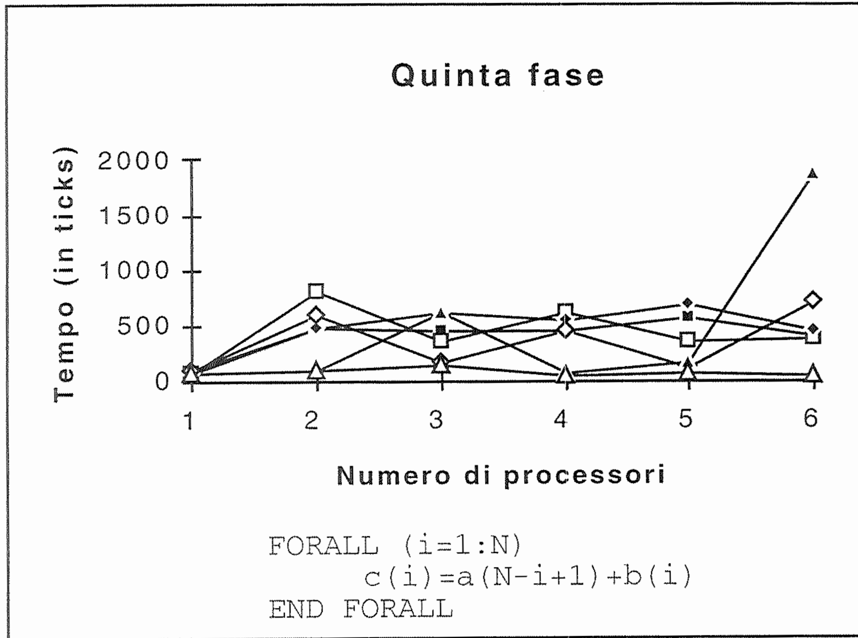

Figura 2.8: Quinta fase - *pghpf* su IBM SP2

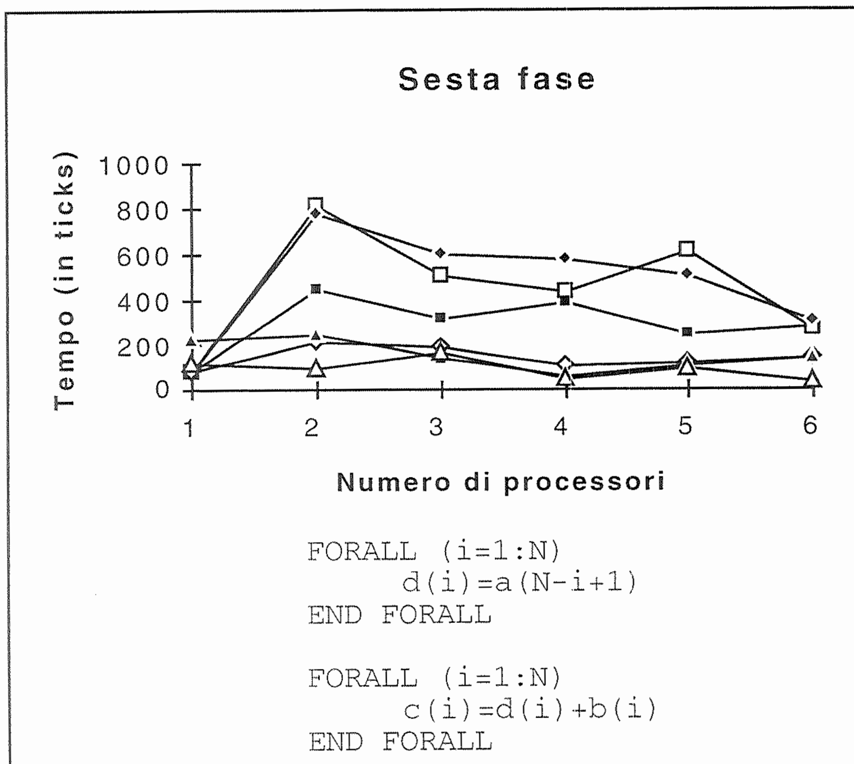

Figura 2.9: Sesta fase - *pghpf* su IBM SP2

I compilatori XL HPF e DEC Fortran 90 non supportano la ridistribuzione dinamica dei dati, quindi non eseguono le fasi settima e ottava. Per tale motivo nelle tabelle non riportiamo

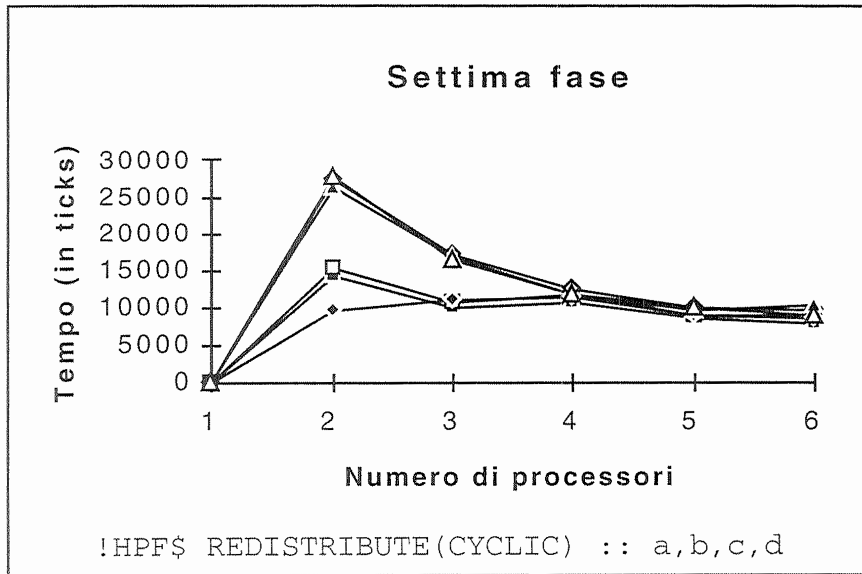

Figura 2.10: Settima fase - *pghpf* su IBM SP2

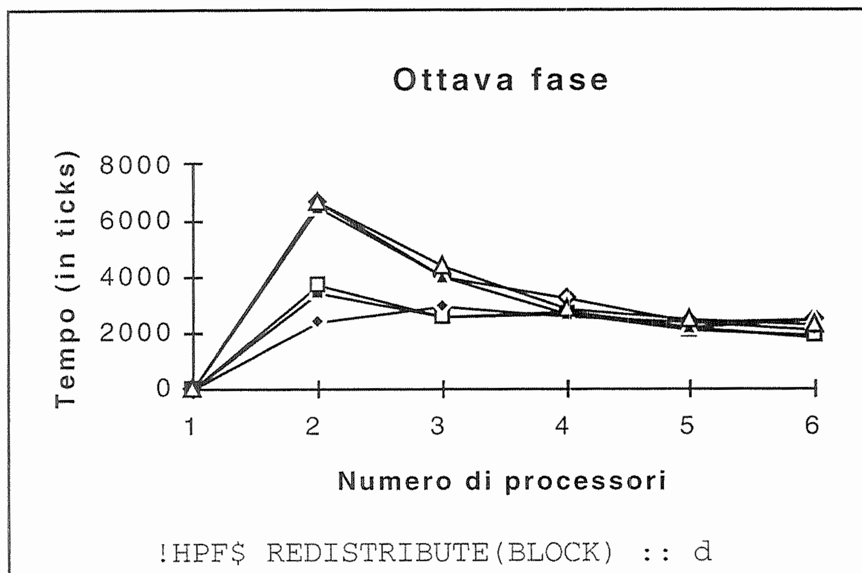

Figura 2.11: Ottava fase - *pghpf* su IBM SP2

lo *speed-up* relativo a queste fasi, sapendo già che in un confronto fra *pghpf* ed ADAPTOR è il primo fra i due che riesce a realizzare un codice scalabile in presenza di ridistribuzioni.

Inoltre il compilatore della IBM ha dei tempi di esecuzione pari quasi ad un tick o poco più per le fasi dalla prima alla quarta, per tale motivo è impossibile calcolare la scalabilità in queste fasi.

Le fasi dalla prima alla quarta non necessitano di comunicazioni per essere eseguite. Il *pghpf* in tali situazioni riesce ad ottenere del codice con scalabilità lineare, sfruttando bene la macchina parallela, mentre ADAPTOR riesce ad ottenere del codice con scalabilità 3 con 8 processori. Il Digital Fortran 90, tranne che per l'esecuzione della prima fase, ha dei tempi

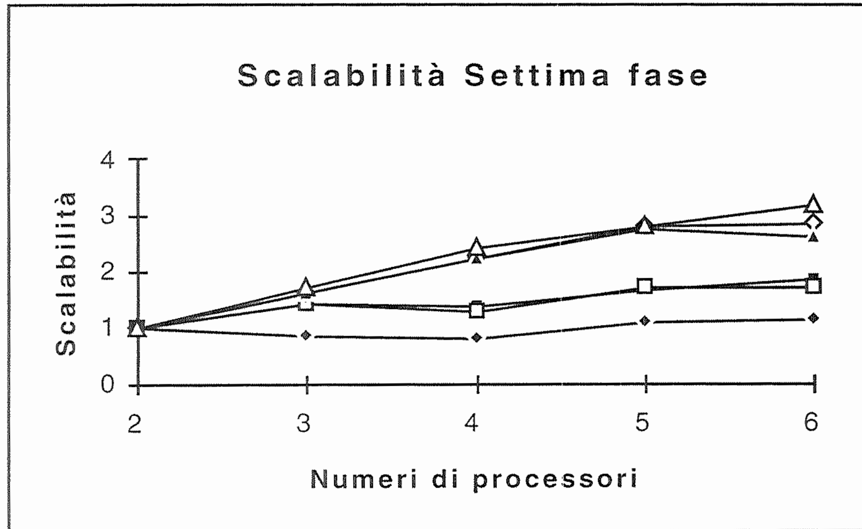

Figura 2.12: Speedup Settima fase rispetto all'esecuzione su due processori - *pghpf* su IBM SP2

	Numero di processori		
	2	*4*	*8*
prima fase	1.24	4.1	8.64
seconda fase	1.8	3.55	6.36
terza fase	1.87	1.1	7.76
quarta fase	1.9	3.4	8.5
quinta fase	0.7	0.7	3
sesta fase	0.8	1.1	3.35
nona fase	0.5	0.37	0.28
decima fase	0.8	1.8	3.65

Tabella 2.1: Scalabilità di Primi_Passi compilato con *pghpf* su Meiko CS-2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori.

di completamento inferiori nel caso sequenziale che non in quello parallelo. Tali risultati possono essere dovuti al fatto che il carico, su ogni nodo della macchina utilizzata, in media si assesta su valori pari al 90-98%. Il compilatore della IBM, che ha lo svantaggio rispetto al *pghpf* ed ADAPTOR di essere disponibile soltanto per macchine IBM Scalable Power, ha dei tempi di esecuzione molto bassi, dovuti sicuramente alla scarsa dimensione computazionale del problema, ma probabilmente anche al fatto che sfrutta bene le caratteristiche della macchina.

Le fasi cinque e sei per la loro esecuzione necessitano di un *pattern* di comunicazioni strutturate. Il compilatore *pghpf* riesce a produrre del codice con scalabilità 3.3 con 8 pro-

	Numero di processori		
	2	*4*	*8*
prima fase	1.9	3.6	5.5
seconda fase	2	3	3
terza fase	2	3	3
quarta fase	2	3	3
quinta fase	0.04	0.05	0.1
sesta fase	0.04	0.06	0.05
nona fase	1.8	3.39	4
decima fase	1.88	3.1	7.5

Tabella 2.2: Scalabilità di Primi_Passi compilato con ADAPTOR su Meiko CS-2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori.

	Numero di processori		
	2	*4*	*8*
quinta fase	0.8	1.4	2.3
sesta fase	1	2.3	2.3
nona fase	0.6	3.75	2.5
decima fase	0.6	0.36	0.17

Tabella 2.3: Scalabilità di Primi_Passi compilato con XL HPF su IBM SP2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori.

cessori, il compilatore della IBM produce del codice poco scalabile. Mentre per ADAPTOR e il Digital Fortran 90 l'esecuzione parallela è più lenta di quella sequenziale.

Gli indirizzamenti indiretti sono gestiti molto bene dal compilatore ADAPTOR, un po' meno dal *pghpf*. Per l'esecuzione della nona fase sono necessarie delle comunicazioni, per ottenere l'indice dell'elemento del vettore a cui fare l'assegnamento. In tale situazione ADAPTOR ottiene una scalabilità pari a 4 con 8 processori ed il compilatore della IBM una scalabilità di 2.5 con 8 processori, mentre con gli altri compilatori è più veloce l'esecuzione su un solo processore. La decima fase, nonostante includa un indirizzamento indiretto, non contiene comunicazioni poiché il vettore f è replicato. Infatti ADAPTOR realizza del codice con scalabilità lineare, mentre il *pghpf* riesce ad ottenere una scalabilità di 3.6 con 8 processori. Stupisce che il compilatore della IBM, nonostante nel manuale si dichiari esplicitamente che oggetti e template non esplicitamente distribuiti saranno replicati (pagina 249 del manuale [IBM96]), non riesce ad ottenere del codice scalabile.

	Numero di processori	
	2	*4*
prima fase	1	1.5
seconda fase	0.8	1.3
terza fase	0.2	0.5
quarta fase	0.5	0.6
quinta fase	0.06	0.12
sesta fase	0.08	0.18
nona fase	0.2	0.18
decima fase	0.19	0.16

Tabella 2.4: Scalabilità di Primi_Passi compilato con DEC Fortran 90 su Digital Alpha System. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori.

2.2 PARKBENCH

Nella *suite* PARKBENCH[1] ([DMS96], [HRV95], [HB94]) sono inclusi alcuni *kernel* per la valutazione di compilatori HPF. Questi semplici programmi sono stati progettati con l'intento di verificare la bontà del compilatore in differenti aree applicative. Vediamo in dettaglio i singoli kernel:

1. Kernel TM - `TEMPLATE`. Questo kernel fornisce esempi di distribuzione "guidata" dall'utente tramite la definizione di opportuni `TEMPLATE` e le relative direttive di allineamento e distribuzione. L'uso adeguato di questi elementi del linguaggio, unito alla capacità specifica del compilatore, dovrebbe permettere un adeguato bilanciamento del carico e una minimizzazione delle comunicazioni.

2. Kernel AA, SH, ST e IR - Comunicazioni. Questi kernel contengono assegnamenti nei quali è necessario accedere ad elementi non locali. Per ottenere una buona scalabilità il compilatore deve essere in grado di riconoscere i *pattern* di comunicazione strutturata e generare il codice appropriato. I kernel AA, SH e ST contengono comunicazioni strutturate, mentre il kernel IR contiene comunicazioni non strutturate.

3. Kernel RD - Intrinseche e libreria HPF. Questo kernel contiene chiamate ad alcune funzioni intrinseche Fortran 90 e di libreria HPF.

4. Kernel AS, IT e IM - Passaggio di array distribuiti come argomenti di procedure. Il passaggio di array distribuiti come argomenti di procedure è forse una delle aree più critiche della programmazione HPF. In questi kernel vengono esplorate alcune situazioni tipiche.

[1] Disponibile agli indirizzi Internet `http://www.netlib.org/parkbench/` e `http://www.npac.syr.edu/users/haupt/parkbench/parkbench.html`

2.2.1 *pghpf* vs ADAPTOR

Questi benchmark sono stati eseguiti sulla Meiko CS-2 utilizzando, come primitive di comunicazione, MPI per il *pghpf* (non essendo supportato Elan) e Elan per ADAPTOR. Nonostante ciò *pghpf* è risultato migliore di ADAPTOR, generando un codice più veloce e più scalabile.

La scalabilità osservata è buona, utilizzando il compilatore *pghpf* con ogni benchmark, tranne per IM ed IT. Nel primo caso la scalabilità è scarsa, mentre nel secondo il tempo di completamento tende ad aumentare all'aumentare del numero dei processori usati. In tabella 2.5 sono riportati i tempi di completamento dei test eseguiti con *pghpf*.

	Numero di processori						
	1	*2*	*4*	*8*	*16*	*32*	*64*
aa	5448	2675	1220	521	219	99	<u>52</u>
as	1199	630	371	258	165	140	<u>115</u>
im	5384	3299	2071	1476	<u>1218</u>	1322	1634
ir	47283	45378	25708	10354	4835	3063	<u>1885</u>
it	2085	<u>1398</u>	1610	1913	1899	2425	3135
rd	7686	3898	1981	985	574	380	<u>256</u>
sh	8445	4389	2474	1081	521	278	<u>143</u>
st	6354	3209	1699	907	457	272	<u>154</u>
tm	7164	3536	1813	893	428	224	<u>114</u>

Tabella 2.5: Tempi di completamento (in ticks) dei *kernel* della *suite* Parkbench con *pghpf* su Meiko CS-2. I tempi di completamento minimi sono sottolineati.

Per i benchmark AA, RD, SH e TM, *pghpf* genera un codice che risulta scalare in maniera lineare se non addirittura super-lineare. Per quanto riguarda AA e TM questo è prevedibile in quanto non ci sono comunicazioni. Invece sono interessanti le prestazioni ottenute da RD, che fa uso di molte funzioni intrinseche quali `MINVAL`, `COUNT`, `SUM` e altre, e da SH che contiene molte comunicazioni e che risulta molto difficile da parallelizzare anche generando a mano il codice *message passing*. Anche IR risulta scalabile nonostante contenga molte comunicazioni non strutturate. Leggermente inferiori sono invece le prestazioni del benchmark ST. Comunque, va considerata l'alta complessità delle comunicazioni implicate dai complessi assegnamenti di questo programma. Infine, decisamente inferiori sono le prestazioni di AS, che contiene solo comunicazioni strutturate di elementi adiacenti. La sua non buona scalabilità può essere dovuta all'uso di procedure con passaggio di parametri distribuiti. Il dubbio che il passaggio di parametri distribuiti possa causare qualche problema al *pghpf* viene rafforzato dall'osservazione delle prestazioni dei benchmark IM e IT che contengono appunto chiamate a procedure con passaggio di parametri distribuiti.

Da notare che non utilizzando l'opzione di compilazione *-Mmpi* il codice generato da *pghpf* risulta assolutamente non scalabile (anzi i tempi di completamento aumentano all'aumentare dei processori), molto più lento e in molti casi non è possibile eseguire il programma con più di 32 processori.

Con gli stessi benchmark ADAPTOR ha avuto un comportamento alquanto strano. In

molti casi non è stato possibile compilare il programma, mentre in altri casi non si sono potute provare le prestazioni con 32 o 64 processori per problemi dovuti all'uso della libreria DALIB. In tabella 2.6 sono riportati i tempi ottenuti dalle esecuzioni dei benchmark. Le prestazioni ottenute da ADAPTOR in questi test sono state decisamente peggiori di quelle ottenute dal *pghpf*, sia per quanto concerne i tempi di completamento che per la scalabilità. In alcuni casi comunque il compilatore si è comportato egregiamente (benchmark ST e SH) dimostrando di essere in grado di trattare adeguatamente istruzioni che richiedano comunicazioni sia strutturate che non strutturate. Assai deludenti sono invece state le prestazioni per test che contengono chiamate a procedure e passaggio di parametri distribuiti.

	Numero di processori						
	1	*2*	*4*	*8*	*16*	*32*	*64*
as	5248	3017	1358	618	326	190	<u>136</u>
ir	954811	539978	280699	145706	107884	<u>63063</u>	81612
it	8962	6816	<u>5815</u>	6179	6888	9105	17900
sh	10028	5175	2631	1319	620	<u>303</u>	-
st	16588	8600	4605	2073	779	323	<u>188</u>
tm	20999	10291	4560	1885	<u>682</u>	-	-

Tabella 2.6: Tempi di completamento (in ticks) dei *kernel* della *suite* Parkbench con ADAPTOR su Meiko CS-2. I tempi di completamento minimi sono sottolineati.

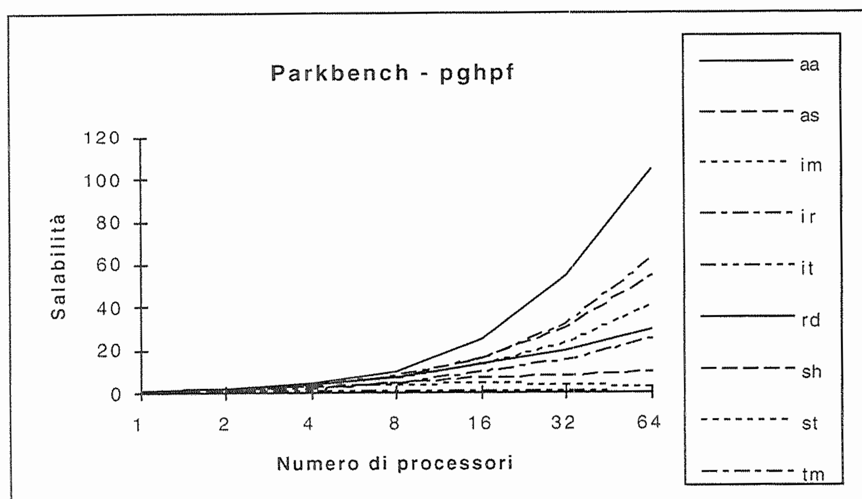

Figura 2.13: Scalabilità di Parkbench - *pghpf*

In figura 2.13 e 2.14 sono riportati i grafici che descrivono la scalabilità di tutti i benchmark della suite PARKBENCH, usando ripettivamente *pghpf* e ADAPTOR.

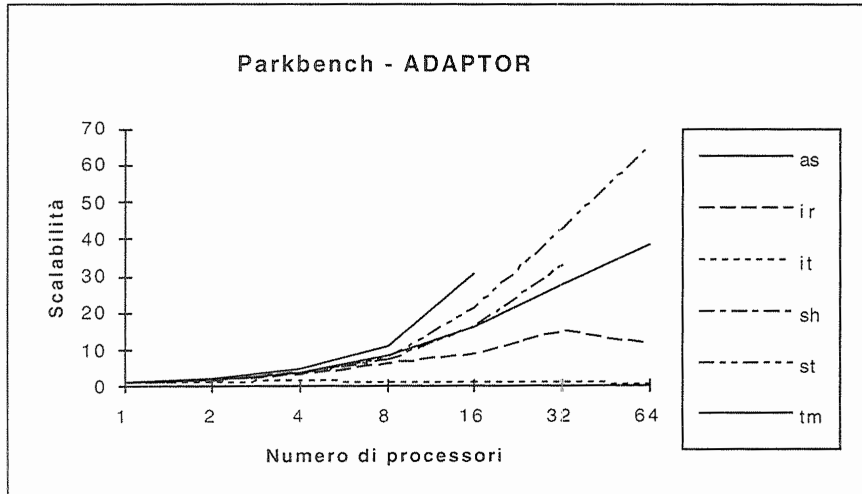

Figura 2.14: Scalabilità di Parkbench - ADAPTOR

2.2.2 Confronto fra tutti i compilatori

Anche con questi *kernel* l'unico possibile confronto che abbiamo potuto effettuare per differenti compilatori su differenti architetture è il confronto della scalabilità.

I compilatori XL HPF, DEC Fortran 90 ed ADAPTOR non compilano il *kernel* IM perché non riconoscono la direttiva

```
CHPF$ DISTRIBUTE X * ONTO *Q
```

che non appartiene al *subset* HPF e prevede che il parametro formale erediti la distribuzione del parametro attuale, qualunque essa sia.

Col compilatore DEC Fortran 90 si riesce a compilare il *kernel* IR solo per l'esecuzione su un processore, ma in fase di esecuzione si ottengono dei valori differenti da quelli attesi. Mentre il *kernel* RD si compila solo per l'esecuzione su 3 e 4 processori, anche tale esecuzione fornisce valori differenti da quelli attesi.

Da un'analisi della tabella 2.7 emerge che il codice generato dal DEC Fortran 90 oltre a non fornire i valori attesi da una determinata computazione molto spesso genera del codice che non giustifica l'esecuzione in parallelo. Tali risultati possono essere dovuti, come già detto in precedenza, al fatto che il carico, su ogni nodo della macchina utilizzata, in media si assesta su valori pari al 90-98%.

Nel *kernel* TM si definisce un template TMPL al quale sono poi allineati dei vettori e delle matrici. Seguendo tali distribuzioni il ciclo `FORALL` centrale non necessita assolutamente di comunicazioni per essere risolto. Per tali motivi sia ADAPTOR che in misura minore il compilatore XL HPF riescono ad ottenere un codice con scalabilità super–lineare, mentre *pghpf* realizza un codice che ha scalabilità lineare. In tale situazione anche il compilatore Digital Fortran 90 realizza un codice con un minimo di scalabilità.

Il *kernel* AA contiene un ciclo `FORALL`, per l'esecuzione del quale non sono necessarie comunicazioni. Questa è quindi la situazione ottima per realizzare del codice scalabile, infatti sia il compilatore XL HPF che il *pghpf* producono un eseguibile con scalabilità super–lineare, mentre il compilatore della Digital produce del codice assolutamente non scalabile.

Per l'esecuzione del *kernel* SH bisogna realizzare una fase di comunicazione, poiché per il calcolo dell'elemento i–esimo è necessario conoscere il valore degli elementi dall'$(i+1)$–esimo

	Numero di processori	
	2	*4*
aa	0.19	0.8
as	-	-
im	-	-
ir	-	-
it	0.03	0.05
rd	-	-
sh	0.45	0.95
st	0.8	2.48
tm	1.08	2.27

Tabella 2.7: Scalabilità dei *kernel* della *suite* Parkbench compilati con DEC Fortran 90 su DEC 2100. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2 e 4 processori. Il trattino indica l'incapacità di compilare il codice.

	Numero di processori		
	2	*4*	*8*
aa	2.03	4.46	10.45
as	1.9	3.23	4.6
im	1.63	2.59	3.64
ir	1.04	1.83	4.56
it	1.49	1.29	1.08
rd	1.97	3.87	7.8
sh	1.92	3.41	7.8
st	1.98	3.73	7
tm	2.02	3.95	8.02

Tabella 2.8: Scalabilità dei *kernel* della *suite* Parkbench compilati con *pghpf* su Meiko CS-2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori.

all'*(i+6)*-esimo. In tale situazione il compilatore XL HPF produce un eseguibile che ha scalabilità super-lineare, mentre il *pghpf* e ADAPTOR producono un codice con scalabilità

	Numero di processori		
	2	*4*	*8*
aa	-	-	-
as	1.73	3.86	8.49
im	-	-	-
ir	1.76	3.4	6.55
it	1.3	1.5	1.4
rd	-	-	-
sh	1.9	3.8	7.6
st	1.9	3.6	8
tm	2.04	4.6	11.14

Tabella 2.9: Scalabilità dei *kernel* della *suite* Parkbench compilati con ADAPTOR su Meiko CS-2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori. Il trattino indica l'incapacità di compilare il codice.

	Numero di processori		
	2	*4*	*8*
aa	1.85	4.3	13
as	1.6	2	2.57
im	-	-	-
ir	0.47	0.34	0.18
it	0.06	0.05	0.06
rd	1.77	2.15	9.41
sh	5.1	15.3	32
st	2.96	4.52	6.16
tm	2.19	4.86	9.33

Tabella 2.10: Scalabilità dei *kernel* della *suite* Parkbench compilati con XL HPF su IBM SP-2. I numeri rappresentano la scalabilità, rispetto all'esecuzione del codice HPF su un processore, dell'esecuzione su 2, 4 e 8 processori. Il trattino indica l'incapacità di compilare il codice.

lineare. Il compilatore DEC Fortran 90 produce un codice che anche in questa situazione è meglio eseguire su un solo processore.

Anche l'esecuzione del *kernel* ST presuppone la realizzazione di una fase di comunicazioni strutturate, in cui il calcolo dell'elemento i–esimo necessita della conoscenza dell'elemento $(2 \times i - 1)$–esimo. L'unico compilatore che riesce a produrre un codice che ha scalabilità lineare è ADAPTOR, mentre il *pghpf* riesce ad ottenere una scalabilità leggermente migliore del compilatore della IBM. Notiamo inoltre che, nonostante le condizioni di carico della macchina, il compilatore della DEC riesce ad ottenere un codice leggermente scalabile.

Il *kernel* IR contiene delle comunicazioni non strutturate con degli indirizzamenti indiretti. In tale situazione il compilatore *pghpf* produce un codice con scalabilità 4.56 con 8 processori, ADAPTOR genera un codice più vicino a quello lineare, mentre XL HPF produce un codice assolutamente non scalabile.

Il *kernel* RD contiene chiamate a funzioni intrinseche non elementari, quali `SUM`, `MINVAL`, `COUNT`, `ANY` ed `ALL`. Il compilatore XL HPF produce un codice con scalabilità super–lineare, mentre il *pghpf* produce un codice con scalabilità lineare.

Il *kernel* AS contiene il passaggio dei parametri alla procedura `SUBA` con direttiva descrittiva, la quale non richiede la ridistribuzione dei parametri attuali all'ingresso della procedura. All'interno di tale procedura è presente un ciclo `FORALL` per eseguire il quale è necessaria la comunicazione fra processori adiacenti. Il compilatore ADAPTOR produce un codice lineare, mentre *pghpf* realizza un codice con scalabilità 4.6 con 8 processori. Il compilatore XL HPF non è in grado di produrre un codice scalabile.

Il *kernel* IT contiene due chiamate alla stessa procedura, in cui è dichiarata la direttiva `INHERIT`. Il compilatore XL HPF in fase di compilazione dichiara di ignorare tale direttiva. Nessun compilatore riesce ad estrarre un codice scalabile.

Il *kernel* IM ha la chiamata ad una subroutine in cui i parametri formali sono distribuiti secondo la distribuzione del parametro attuale. Questa è una caratteristica del linguaggio non in *subset* HPF, solo *pghpf* compila questo codice, realizzando un eseguibile con scalabilità 3.6 con 8 processori.

Da ciò si evince che XL HPF in caso di comunicazione strutturate tende a realizzare un eseguibile con scalabilità più elevata rispetto a *pghpf* ed ADAPTOR, mentre in presenza di indirizzamenti indiretti questi ultimi due riescono a sfruttare il parallelismo contrariamente al compilatore della IBM. Nel caso di passaggio dei parametri distribuiti a procedure solo il *pghpf* realizza un codice in parte scalabile. Per quanto riguarda l'implementazione delle procedure intrinseche utilizzate nel *kernel* RD sia *pghpf* che XL HPF realizzano un codice scalabile, anche super–lineare nel caso del compilatore della IBM, probabilmente avvantaggiato nella sua realizzazione da una profonda conoscenza delle caratteristiche della macchina IBM SP2.

2.3 Ulteriore confronto fra *pghpf* ed ADAPTOR

2.3.1 Purdue Set

Insieme ad ADAPTOR, il GMD fornisce una serie di benchmark per HPF. Tra questi abbiamo scelto quelli appartenenti al Purdue Set (per i listati vedere l'appendice A.3 a pagina 45), per la varietà di problematiche che affrontano. Si tratta di 14 problemi realizzati alla Purdue University da John R. Rice e riadattati per il FORTRAN D da T. Haupt al Northeast Parallel Architectures Center della Syracuse University. In tutti i programmi di questo insieme di benchmark non si fa assolutamente uso di direttive per la distribuzione dei dati, e quindi si lasciano ampie possibilità di scelta al compilatore.

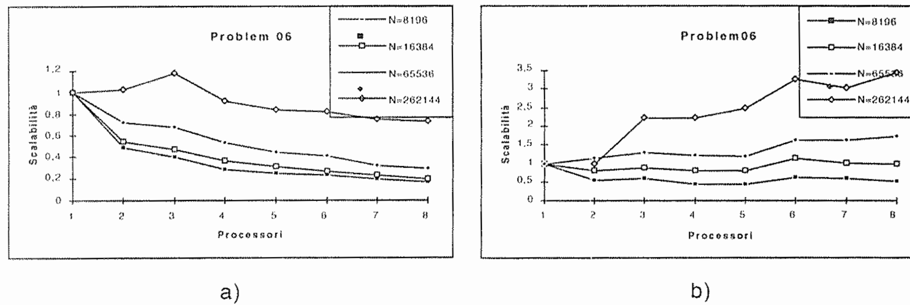

Figura 2.15: Scalabilità del Problem06 con ADAPTOR (a) e *pghpf* (b)

2.3.2 Risultati delle prove

Anche questi benchmark sono stati eseguiti sull'architettura IBM SP2. I risultati di ADAPTOR e *pghpf* hanno, in molti casi, lo stesso andamento per quanto riguarda la scalabilità. Per i programmi dal Problem02 al Problem05 (per i listati vedere le appendici dalla A.3.1 alla A.3.5) il volume di comunicazioni è basso e quindi entrambi scalano bene quando sono elaborati molti dati, mentre vanno in saturazione per problemi con pochi dati. Il Problem01 invece non ha una buona scalabilità a causa dell'uso della funzione intrinseca `SUM`, usata per la somma degli elementi di un vettore distribuito, che si è dimostrata inefficiente anche in altre prove. Il Problem06 (per il listato vedere l'appendice A.3.6 a pagina 51) introduce delle comunicazioni strutturate e qui si nota una differenza tra i due compilatori. Mentre *pghpf* mostra un lieve abbassamento del tempo di completamento all'aumentare dei processori (scalabilità di 3.44 con 8 processori, vedi figura 2.15b), ADAPTOR ha una tendenza opposta (figura 2.15a). Ancora una volta si deve constatare come ADAPTOR non sia in grado di gestire adeguatamente comunicazioni strutturate. Questa ipotesi è avvalorata dal Problem07 (il cui listato si trova nel paragrafo A.3.7 a pagina 53) in cui si fa un uso intensivo delle funzioni intrinseche Fortran 90 come `SPREAD`, `PRODUCT` e `SUM`. Questo mostra un'accettabile scalabilità se compilato con *pghpf*, mentre con ADAPTOR la scalabilità è pessima. In questo problema si ha un andamento piuttosto curioso (vedi fig. 2.16) che è forse dovuto all'implementazione delle funzioni intrinseche Fortran 90. Nei tempi ottenuti dall'esecuzione di questo programma si sono infatti ottenute notevoli differenze tra l'esecuzione con un numero pari e con un numero dispari di processori. Nell'esecuzione su d processori, con d numero dispari, si sono ottenuti tempi sempre minori dell'esecuzione su $d-1$ e $d+1$ processori.

Per quanto riguarda i rimanenti problemi nessuno scala in maniera accettabile, tranne il Problem15 per il quale però si ottiene il tempo minimo con l'uso di soli 3 processori, dopo di che il tempo di completamento tende ad aumentare all'aumentare del numero di processori.

Tutti i test visti in precedenza sono stati effettuati sulla IBM SP2 senza abilitare l'opzione di ottimizzazione del compilatore *pghpf*. Questi stessi test sono poi stati ripetuti sulla Meiko CS-2 utilizzando il parametro `-O2` per abilitare le ottimizzazioni in fase di compilazione. I risultati ottenuti sono sicuramente migliori e denotano che tutti gli algoritmi dal Problem02 al Problem08 scalano pressoché perfettamente. Un esempio dell'andamento del tempo di completamento del Problem02 è mostrato in figura 2.17. Stupisce soprattutto il caso del Problem08 che sull'SP2 (senza ottimizzazioni) mostrava un aumento dei tempi di esecuzione all'aumentare del numero di processori, mentre abilitando le ottimizzazioni il codice risultante presenta una buona scalabilità (8,073 con 16 processori, vedi figura 2.19). In figura 2.18a e 2.18b sono mostrati i grafici relativi al tempo di esecuzione del Problem08 compilato rispettivamente con *pghpf* su IBM SP2 senza ottimizzazioni e con *pghpf* su Meiko

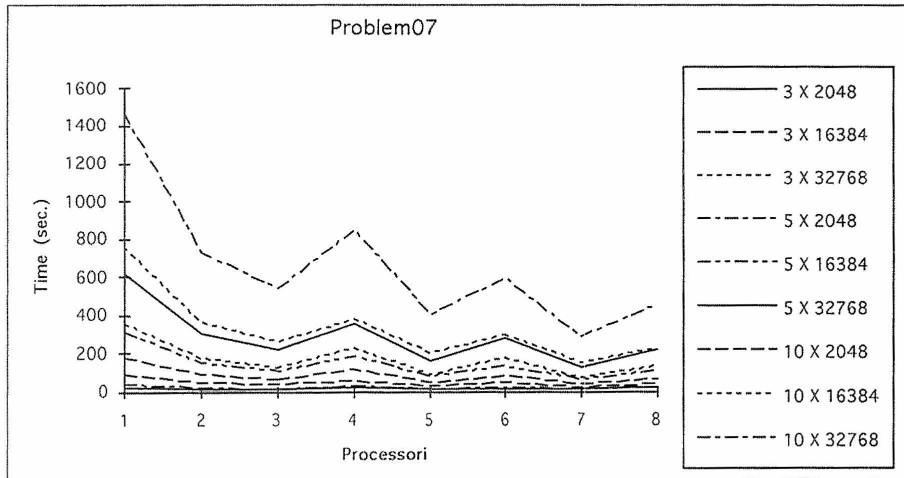

Figura 2.16: Tempi di esecuzione del Problem07 al variare del numero di processori e della dimensione del problema

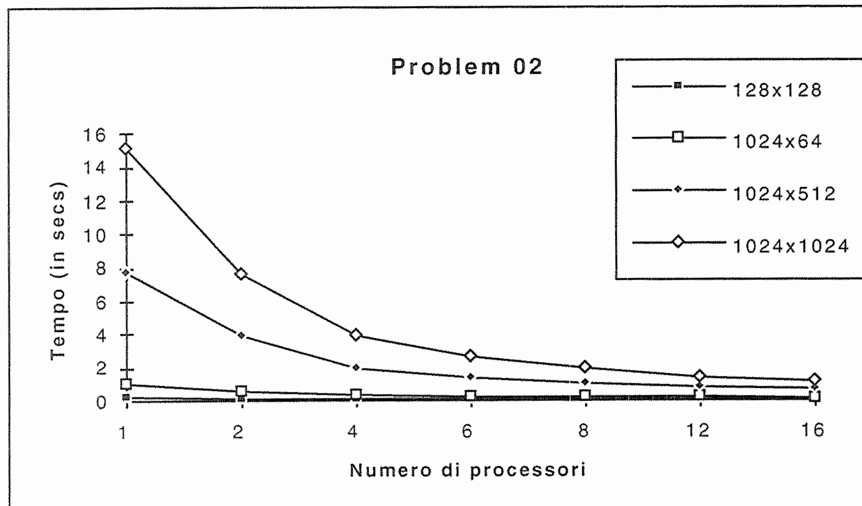

Figura 2.17: Tempi di completamento di Problem02 compilato con *pghpf* su Meiko CS-2

CS-2 con le ottimizzazioni abilitate. come mostrato in figura 2.20 per il Problem15, il tempo di completamento diminuisce utilizzando fino a 8 processori con volume dei dati in ingresso di 8 192 unità, e sino a 12 con 16 384 dati in ingresso, dopo di che il tempo di completamento tende ad aumentare. Anche qui i miglioramenti rispetto alla versione non ottimizzata sono evidenti considerando che in precedenza si aveva scalabilità solo sino a 3/4 processori. In fine i programmi da Problem09 (vedi figura 2.21) a Problem14 non scalano affatto a causa delle complesse fasi di riduzione e delle molte comunicazioni necessarie.

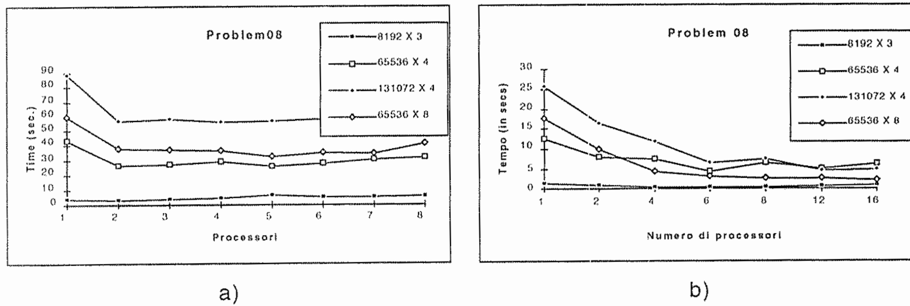

Figura 2.18: Tempi di completamento di Problem08 compilato con *pghpf* su IBM SP2 (a) e su Meiko CS-2 (b)

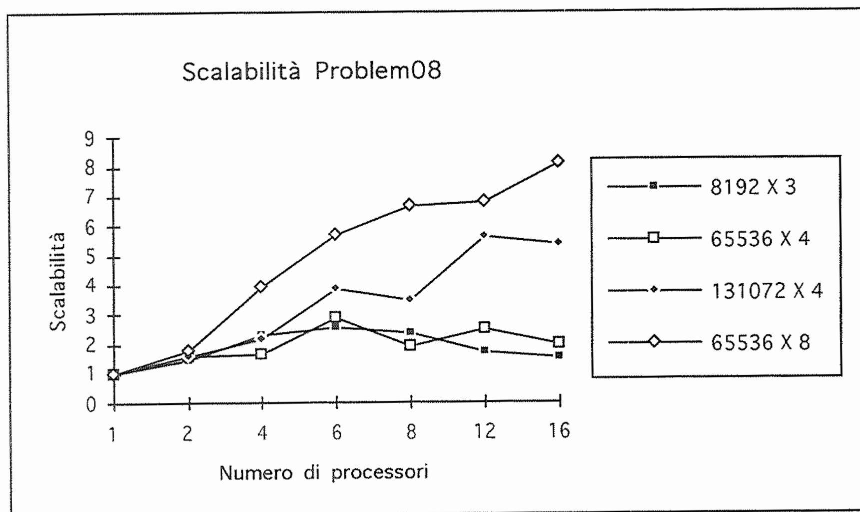

Figura 2.19: Scalabilità di Problem08 compilato con *pghpf* su Meiko CS-2

2.4 Conclusioni

I risultati ottenuti in tutti i benchmark eseguiti ci fanno dedurre che *pghpf* sia il compilatore migliore, perché è l'unico che rispetta le specifiche sintattiche del HPF Language Specification 1.1 ([For94]), che compila tutti i benchmark proposti e che ha degli andamenti costanti. Infatti per problemi che si risolvono senza comunicazioni il compilatore XL HPF riesce a realizzare un eseguibile che ha scalabilità maggiore rispetto a quella del *pghpf* , che è lineare, mentre per problemi che necessitano di *pattern* di comunicazioni strutturate XL HPF produce un eseguibile che non scala, contrariamente al *pghpf* , il cui eseguibile prodotto ha spesso scalabilità lineare.

Per quanto riguarda l'implementazione delle procedure intrinseche e di libreria abbiamo visto che sia *pghpf* che XL HPF realizzano un eseguibile scalabile, anche super–lineare nel caso del compilatore della IBM, ciò è forse dovuto al fatto che quest'ultimo è stato realizzato potendo sfruttare tutte le caratteristiche della macchina IBM SP2.

Nel caso dell'indirizzamento indiretto è ADAPTOR il compilatore che più di tutti riesce a realizzare degli eseguibili scalabili. ADAPTOR può quindi essere considerato un ottimo prodotto (soprattutto considerando il fatto che si tratta di uno strumento *freeware* di cui

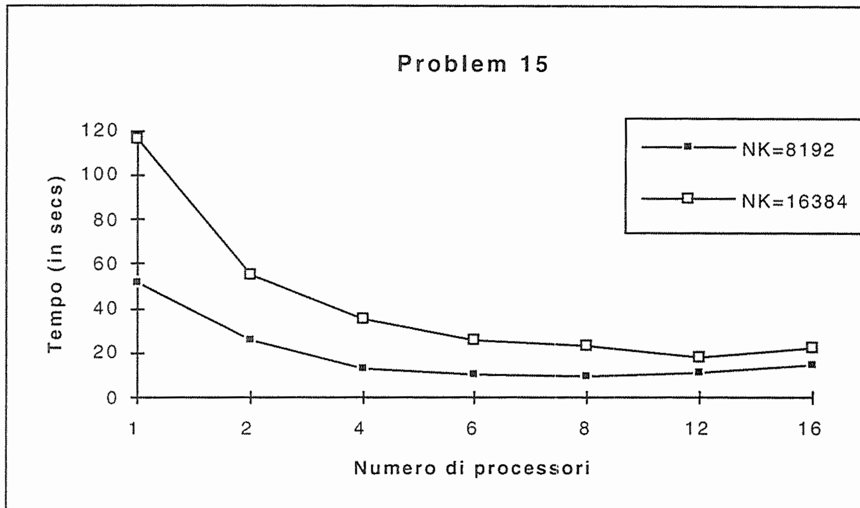

Figura 2.20: Tempi di completamento di Problem15 compilato con *pghpf* su su Meiko CS-2

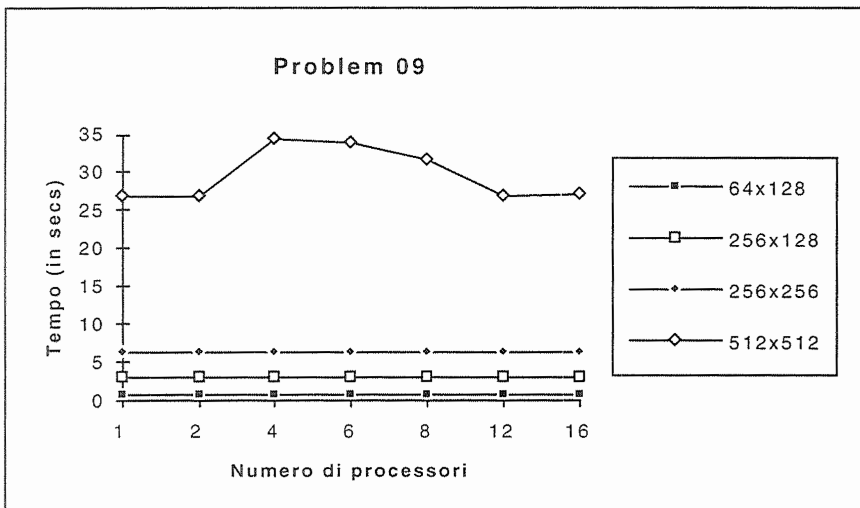

Figura 2.21: Tempi di completamento di Problem09 compilato con *pghpf* su su Meiko CS-2

è disponibile anche il codice sorgente) per l'utilizzo di HPF per scopi didattici e per la parallelizzazione di problemi semplici. Sicuramente utili sono le molte estensioni presenti, di cui alcune già approvate per HPF 2.0.

Il manuali forniti con i compilatori si sono dimostrati insufficienti in molti punti e completamente privi di suggerimenti, creando non pochi problemi nella fase iniziale del loro utilizzo. Dai manuali, soprattutto quelli della DEC e della IBM, è completamente assente una descrizione del meccanismo delle procedure estrinseche, mentre questo è trattato, seppur in modo superficiale, nel manuale del *pghpf* , in cui l'introduzione dell'interfaccia `F77_LOCAL` è sicuramente molto utile per permettere l'uso di codice FORTRAN 77 anche già esistente.

Appendice A

Listati dei programmi

A.1 Primi Passi

```
PROGRAM Primipassi

        INTEGER, PARAMETER :: N=100000
        REAL, DIMENSION(N) :: a,b,c,d
!HPF$   DYNAMIC :: a,b,c,d
!HPF$   PROCESSORS SP(NUMBER_OF_PROCESSORS())
!HPF$   DISTRIBUTE (BLOCK) ONTO SP:: a,b,c,d
        INTEGER i,tp,ta,tr

        PRINT *,"Numero di processori: ",NUMBER_OF_PROCESSORS()

c       Prima fase
        CALL system_clock(tp,tr)
        FORALL(i=1:N)
            a(i)=i
            b(i)=i
          c(i)=i
        END FORALL
        CALL system_clock(ta)
        PRINT *,"Prima fase: ",(ta-tp)

c       Seconda fase
        CALL system_clock(tp,tr)
        FORALL(i=1:N)
          c(i)=a(i)+b(i)
        END FORALL
        CALL system_clock(ta)
        PRINT *,"Seconda fase: ",(ta-tp)

c       Terza fase
        CALL system_clock(tp,tr)
!HPF$   INDEPENDENT
        DO i=1,N
            c(i)=a(i)+b(i)
        END DO
        CALL system_clock(ta)
        PRINT *,"Terza fase: ",(ta-tp)

c       Quarta fase
        CALL system_clock(tp,tr)
        c(1:N)=a(1:N)+b(1:N)
        CALL system_clock(ta)
        PRINT *,"Quarta fase: ",(ta-tp)

c       Quinta fase
        CALL system_clock(tp,tr)
        FORALL (i=1:N)
          c(i)=a(N-i+1)+b(i)
        END FORALL
        CALL system_clock(ta)
        PRINT *,"Quinta fase: ",(ta-tp)
```

```
c       Sesta fase
        CALL system_clock(tp,tr)
        FORALL (i=1:N)
          d(i)=a(N-i+1)
        END FORALL
        FORALL (i=1:N)
          c(i)=d(i)+b(i)
        END FORALL
        CALL system_clock(ta)
        PRINT *,"Sesta fase: ",(ta-tp)

c       Settima fase
        CALL system_clock(tp,tr)
!HPF$   REDISTRIBUTE(CYCLIC) :: a,b,c,d
        CALL system_clock(ta)
        PRINT *,"Settima fase: ",(ta-tp)

c       Ottava fase
        CALL system_clock(tp,tr)
!HPF$   REDISTRIBUTE(BLOCK) :: d
        CALL system_clock(ta)
        PRINT *,"Ottava fase: ",(ta-tp)

        FORALL (i=1:N)
          e(i)=i
          f(i)=i
        ENDFORALL
!HPF$   REDISTRIBUTE(BLOCK) :: a,b,c

c       Nona fase
        CALL system_clock(tp,tr)
        a(e(1:N))=c(1:N)
        CALL system_clock(ta)
        PRINT *,"Nona fase: ",(ta-tp)

c       Decima fase
        CALL system_clock(tp,tr)
        a(f(1:N))=c(1:N)
        CALL system_clock(ta)
        PRINT *,"Decima fase: ",(ta-tp)

        END PROGRAM prova
```

A.2 PARKBENCH

Questi becnhmark sono stati leggermente modificati per permettere l'esecuzione su un numero arbitrario di processori (la versione originale permetteva la sola esecuzione con 8 processori) e la stampa del tempo impiegato per l'elaborazione.

A.2.1 aa.hpf

```
          program AA
C         ======================================================
C         array assignments kernel
C         taken from the Livermore Loops, kernel 9
C         ======================================================
C         please send comments to haupt@npac.syr.edu

          parameter ( N = 100000 )
          real PX(13,N), Q
          real DM22, DM23, DM24, DM25, DM26, DM27, DM28, C0
          integer i, k, it, nit
C       variabili per il timer
          integer tp, ta, tr
          data nit /23/
CHPF$     processors p(NUMBER_OF_PROCESSORS())
CHPF$     template d(N)
```

```
CHPF$     distribute d(block) onto p
CHPF$     align PX(*,I) with d(I)

          print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
C          start_timer
          CALL system_clock(tp,tr)


C      Initialise the arrays and scalars
           DM22 = 1.0
           DM23 = 2.5
           DM24 = 0.11
           DM25 = 0.25
           DM26 = 2.6
           DM27 = 0.2
           DM28 = 10.0
           CO   = 0.5
           FORALL(I=1:13,J=1:N) PX(I,J) = I+1.E-07*J

           DO IT=1,NIT
C***********************************************************************
C***  KERNEL 9      INTEGRATE PREDICTORS
C***********************************************************************
C
C
    FORALL ( i = 1:N )
     *  PX(1,i) =      DM28 * PX(13,i) + DM27 * PX(12,i) +
     *                 DM26 * PX(11,i) + DM25 * PX(10,i) +
     *                 DM24 * PX(9,i)  + DM23 * PX(8,i)  +
     *                 DM22 * PX(7,i)  + CO * (PX(5,i)  +
     *                 PX(6,i))        + PX(3,i)
C
C***********************************************************************
          ENDDO
C
C         stop_timer
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'         PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'         AA  KERNEL'
      print *,' PROBLEM SIZE: N =', N
      print *,' FINAL RESULT = ', PX(1,N/2)
      print *,' EXPECTED VALUE FOR SIZE 1000000'
      print *,' IS  200.083 +/- 0.001'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      STOP
      END
```

A.2.2 as2.hpf

```
      PROGRAM AS
C     ==========================================================
C     assertions on the mapping of the actual argument
C     ==========================================================
C     please send comments to haupt@npac.syr.edu

      PARAMETER(NDIM=512,MDIM=512)
      REAL, DIMENSION(NDIM,MDIM) :: A
C      variabili per il timer
       integer tp, ta, tr
CHPF$ PROCESSORS P(NUMBER_OF_PROCESSORS()/2,2)
CHPF$ DISTRIBUTE A(BLOCK,BLOCK) ONTO P
       INTERFACE
```

```
        SUBROUTINE SUBA(X,N,M)
        REAL, DIMENSION(:,:) :: X
        INTEGER, INTENT(IN) :: N,M
        END SUBROUTINE SUBA
      END INTERFACE
      DATA NIT /5/

      print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c     start_timer
      CALL system_clock(tp,tr)

      FORALL(I=2:NDIM-1,J=1:MDIM) A(I,J)=100
      FORALL(J=1:MDIM) A(1,J) = 0
      FORALL(J=1:MDIM) A(NDIM,J) = 0

      DO I=1,NIT
       CALL SUBA(A,NDIM,MDIM)
      ENDDO

c      stop_timer
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'          PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'          AS  KERNEL'
      print *,' PROBLEM SIZE: N, M =', NDIM,MDIM
      print *,  NIT,' ITERATIONS'
      print *,' FINAL RESULT = ', A(NDIM-2,2)
      print *,' EXPECTED VALUE FOR SIZE 512 BY 512'
      print *,' AFTER 5 ITERATIONS'
      print *,' IS  85.547 +/- 0.001'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      STOP
      END

      SUBROUTINE SUBA(X,N,M)
      REAL, DIMENSION(:,:) :: X
        INTEGER, INTENT(IN) :: N,M
CHPF$ PROCESSORS Q(NUMBER_OF_PROCESSORS()/2,2)
CHPF$ DISTRIBUTE X *(BLOCK,BLOCK) ONTO *Q
C     ============================================================
C     Array X is asserted to already be distributed (BLOCK,BLOCK)
C     onto Q so, if possible, no data movement should occur
C     ============================================================

      FORALL (I=2:N-1,J=2:M-1) X(I,J)=0.25*(X(I+1,J)+X(I-1,J)+
     &        X(I,J-1)+X(I,J+1))

      RETURN
      END
```

A.2.3 im.hpf

```
      PROGRAM IM
C     ==========================================================
C     inherted mapping (see comments to subroutine suba)
C     addresses features not included in Subset HPF
C     ==========================================================
C     please send comments to haupt@npac.syr.edu

C      variabili per il timer
      integer tp, ta, tr
```

```
      PARAMETER (N=4096)
      REAL, DIMENSION(N) :: A,B
      INTERFACE
       SUBROUTINE SUBA(X)
       REAL, DIMENSION(:) :: X
CHPF$  PROCESSORS Q(NUMBER_OF_PROCESSORS())
CHPF$  DISTRIBUTE X * ONTO *Q
       END SUBROUTINE SUBA
      END INTERFACE
CHPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
CHPF$ DISTRIBUTE A(BLOCK) ONTO P
CHPF$ DISTRIBUTE B(CYCLIC) ONTO P

      LOGICAL DEBUG
c      DATA DEBUG /.FALSE./
      DATA DEBUG /.TRUE./

      print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c         start_timer()
      CALL system_clock(tp,tr)

      FORALL(i=1:N) A(I)=I
      FORALL(i=1:N) B(I)=I

      DO I=1,400
       CALL SUBA(A)
       CALL SUBA(B)
      ENDDO
      fresult=b(1)
c         stop_timer()
      CALL system_clock(ta)

      print *,' '
      print *,'*******************************'
      print *,'         PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'         IM  KERNEL'
      print *,' PROBLEM SIZE: N  =', N
      print *,' FINAL RESULT = ',fresult
      print *,' EXPECTED VALUE FOR ANY N'
      print *,'      is 1.0 +/- 0.00001'
      print *,'*******************************'

c       print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      IF(DEBUG) THEN
       ierror=0
       if(any(abs(a-b).gt.0.0001)) then
        print *,'wrong result'
       else
        print *,' A is equal to B, as expected'
       endif
      ENDIF

      END

      SUBROUTINE SUBA(X)
      REAL, DIMENSION(:) :: X
      REAL W (SIZE(X))
CHPF$ PROCESSORS Q(NUMBER_OF_PROCESSORS())
CHPF$ DISTRIBUTE X * ONTO *Q
CHPF$ ALIGN W(:) WITH X(:)
C     -----------------------------------------------------
C     compiler, use whatever distribution format the actual
C     argument had so, if possible, no data movement should
C     occur [this feature is not defined in Subset HPF]
C     X is asserted to be already distributed onto Q
C     ------------------------------------------------------
      W=CSHIFT(X,1)
```

```
      X=W-1
      RETURN
      END
```

A.2.4 ir.hpf

```
      PROGRAM IR
C     ============================================================
C     irregular communications kernel
C     ============================================================
C     please send comments to haupt@npac.syr.edu

      INTEGER edges
      PARAMETER (M=256)
      PARAMETER (ME = M*(M-1))
      PARAMETER (ME2 = 2* ME)
      PARAMETER (M2 = M*M)
      REAL, DIMENSION(ME2) :: Y
      INTEGER, DIMENSION(ME2) :: L,R,U,D,IM
C      variabili per il timer
      integer tp, ta, tr

CHPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
CHPF$ TEMPLATE T(ME2)
CHPF$ DISTRIBUTE T(BLOCK) ONTO P
CHPF$ ALIGN (:) WITH T(:) :: L,R,U,D,IM,Y

       NIT=250

c       initialization of an unstructured mesh
       FORALL (I=1:ME) IM(I)= (I+I/M)/M
       FORALL(I=1:M2) Y(I)= I/M*0.1 + MOD(M,I)*0.2
       FORALL (I=1:ME) L(I)=I+IM(I)
       FORALL (I=1:ME) R(I)=L(I)+1
       FORALL (I=1:M2-M) U(I)=I
       FORALL (I=1:M2-M) D(I)=I+M
       L(ME+1:ME2) = U(1:ME)
       R(ME+1:ME2) = D(1:ME)

       print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c       start_timer
       CALL system_clock(tp,tr)

       DO IT = 1, NIT
       FORALL(I=1:ME2) Y(L(I)) = 0.5 * (Y(L(I)) + Y(R(I)))
       ENDDO

c      stop_timer
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'         PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'         IR  KERNEL'
      print *,' PROBLEM SIZE: N =', M
      print *,  NIT,' ITERATIONS'
      print *,' FINAL RESULT = ', Y(L(1))
      print *,' EXPECTED VALUE FOR SIZE 200'
      print *,' AFTER 85 ITERATIONS IS'
      print *,'     67.700 +/- 0.001'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)


      STOP
      END
```

A.2.5 it.hpf

```
      PROGRAM IT
C     ==========================================================
C     inherited template
C     ==========================================================
C     please send comments to haupt@npac.syr.edu

      PARAMETER (N=4096)
      PARAMETER (N1 =1, N2 = N/2, N3 = N2+1, N4 = N)
      PARAMETER (JMAX = 1250)
      REAL, DIMENSION(N) :: A
C      variabili per il timer
      integer tp, ta, tr

CHPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
CHPF$ TEMPLATE T(N)
CHPF$ DISTRIBUTE T(BLOCK) onto P
CHPF$ ALIGN A(:) WITH T(:)
      INTERFACE
        SUBROUTINE SUBA(X,N1)
        REAL, DIMENSION(:) :: X
        INTEGER, INTENT(IN) :: N1
CHPF$   INHERIT X
CHPF$   PROCESSORS Q(NUMBER_OF_PROCESSORS())
CHPF$   DISTRIBUTE X *(BLOCK) ONTO *Q
        END SUBROUTINE SUBA
      END INTERFACE

      LOGICAL DEBUG
c      DATA DEBUG /.FALSE./
      DATA DEBUG /.TRUE./

      print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c      start_timer
      CALL system_clock(tp,tr)

      A=50.0
      A(1)=100.0
      A(N)=100.0

      DO J=1,JMAX
       CALL SUBA(A(N1:N2),N2)
       CALL SUBA(A(N3:N4),N2)
      ENDDO

c      stop_timer()
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'        PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'        IT  KERNEL'
      print *,' PROBLEM SIZE: N =', N
      print *,  JMAX,' ITERATIONS'
      print *,' FINAL RESULT = ', A(N-1)
      print *,' EXPECTED VALUE FOR SIZE 4096'
      print *,' AFTER 1250 ITERATIONS IS'
      print *,'     98.872 +/- 0.001'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      IF(DEBUG) THEN
      A=50
      A(1)=100.0
```

```
      A(N)=100.0
      DO J=1,JMAX
        FORALL (I=N1+1:N2-1) A(I)=0.5*(A(I+1)+A(I-1))
        FORALL (I=N3+1:N4-1) A(I)=0.5*(A(I+1)+A(I-1))
      ENDDO
      write(6,*) a(N-1)
      ENDIF
      STOP
      END

      SUBROUTINE SUBA(X,N1)
      REAL, DIMENSION(:) :: X
      INTEGER, INTENT(IN) :: N1
CHPF$ INHERIT X
CHPF$ PROCESSORS Q(NUMBER_OF_PROCESSORS())
CHPF$ DISTRIBUTE X *(BLOCK) ONTO *Q

      FORALL (I=2:N1-1) X(I)=0.5*(X(I+1)+X(I-1))

      RETURN
      END
```

A.2.6 rd.hpf

```
      PROGRAM RD
C     ======================================================
C     reduction functions
C     adopted from the Purdue Set (J. Rice)
C     ======================================================
C     please send comments to haupt@npac.syr.edu

      INTEGER NS,NT
      PARAMETER (NS = 128, NT = 4092 )
C      variabili per il timer
      integer tp, ta, tr
      REAL, DIMENSION(NT,NS) :: SCORES
      LOGICAL, DIMENSION(NT,NS) :: ABOVE

      INTEGER NABOVE
      REAL AVER,AVERTOP,LOWABO
      LOGICAL GENIUS,DEBUG
      DATA DEBUG /.FALSE./
C      DATA DEBUG /.TRUE./

CHPF$ PROCESSORS Q(NUMBER_OF_PROCESSORS())
CHPF$ TEMPLATE TMPL(NT)
CHPF$ DISTRIBUTE TMPL(BLOCK) ONTO Q
CHPF$ ALIGN SCORES(i,*) with TMPL(i)
CHPF$ ALIGN ABOVE(i,*) with TMPL(i)

       print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c      start_timer
       CALL system_clock(tp,tr)

       do it=1,2

        FORALL(I=1:NT,J=1:NS) SCORES(I,J)=0.0006321*I*J
        SCORES=60.0+40.0*SIN(SCORES)

        SSUM=SUM(SCORES)
         AVER=SSUM/(NS*NT)
         WHERE(SCORES.GT.AVER)
           ABOVE=.TRUE.
          SCORES=SCORES*1.1
         ELSEWHERE
          ABOVE=.FALSE.
         ENDWHERE
```

```
          NABOVE=COUNT(ABOVE)
          AVERTOP=SUM(SCORES,MASK=ABOVE)/NABOVE
          LOWABO=MINVAL(SCORES,MASK=ABOVE)

          GENIUS=ANY(ALL(ABOVE,DIM=1))

        enddo
c       stop_timer
       CALL system_clock(ta)

       print *,' '
       print *,'******************************'
       print *,'         PARKBENCH '
       print *,' LOW LEVEL HPF BENCHMARK SUITE'
       print *,'         RD  KERNEL'
       print *,' PROBLEM SIZE: N, M =',NS,NT
       print *,' FINAL RESULT = ',avertop
       print *,' EXPECTED VALUE FOR 128 by 4092'
       print *,'      is 94.396 +/- 0.001'
       print *,'******************************'

       IF(DEBUG) THEN

       PRINT 60, NS,NT
  60   FORMAT ('PROBLEM SIZE',I6,' STUDENTS AND ',I6,' TESTS')
       PRINT *,'AVERAGE TEST SCORE .....:',AVER
       PRINT *,'# SCORES ABOVE AVERAGE..:',NABOVE
       PRINT *,'AVERAGE ABOVE ..........:',AVERTOP
       PRINT *,'LOWEST SCORE ABOVE .....:',LOWABO
       PRINT *,'THERE IS GENIUS ........:',GENIUS
       PRINT *,'--------------------------------------------------'
       PRINT 70, NS,NT
  70   FORMAT ('RESULT WITH ',I6,' STUDENTS AND ',I6,
     +         ' SHOULD BE:')
       PRINT *,'AVERAGE TEST SCORE .....: 60.7349'
       PRINT *,'# SCORES ABOVE AVERAGE..: 266180'
       PRINT *,'AVERAGE ABOVE ..........: 94.3955'
       PRINT *,'LOWEST SCORE ABOVE .....: 66.8090'
       PRINT *,'THERE IS GENIUS ........: FALSE'
       ENDIF

c      print execution time
       print *,'Tempo d''esecuzione:', (ta-tp)
       STOP
       END
```

A.2.7 sh.hpf

```
        program SH
C       ================================================================
C       array assignments with shifts
C       taken from the Livermore Loops, kernel 7
C       ================================================================
C       please send comments to haupt@npac.syr.edu

        parameter ( N = 1000000)
        real, dimension(N) :: X, Y, Z, U
        real Q, R, T
        integer k
C       variabili per il timer
         integer tp, ta, tr

CHPF$    processors p(NUMBER_OF_PROCESSORS())
CHPF$    template d(1000000)
CHPF$    distribute d(block) onto p
CHPF$    align (:) with d(:) :: X, Y, Z, U

         print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c      start_timer
```

```
        CALL system_clock(tp,tr)

C            Initialisation of scalars
           Q = 10.0
        R = 1.5
        T = 3.77
C          Initialization loop
        U = 7.0
           X = 8.0
           Y = 9.0
           Z = 10.0

           do it=1,5
C***********************************************************************
C***  KERNEL 7      EQUATION OF STATE FRAGMENT
C***********************************************************************
C
C
    forall (k=1:N-6)
     &   X(k)=     U(k) + R*( Z(k) + R*Y(k)) +
     &        T*( U(k+3) + R*( U(k+2) + R*U(k+1)) +
     &        T*( U(k+6) + Q*( U(k+5) + Q*U(k+4))))
C
C
C***********************************************************************
          enddo
c         stop_timer
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'         PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'         AS  KERNEL'
      print *,' PROBLEM SIZE: N =', N
      print *,' FINAL RESULT = ', X(N-6)
      print *,' EXPECTED VALUE FOR SIZE 1000000'
      print *,' IS  11211.0 +/- 0.1'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      STOP
      END
```

A.2.8 st.hpf

```
       program ST
C      ==========================================================
C      array section with strides
C      ==========================================================
C      please send comments to haupt@npac.syr.edu


       integer N,M,K1,K2,I
       parameter ( N = 2048)
       parameter ( M = N/2 )
       real, dimension(N,N) :: A, B
       logical debug
C      variabili per il timer
       integer tp, ta, tr
       data debug /.false./
c       data debug /.true./

CHPF$  processors p(NUMBER_OF_PROCESSORS()/2,2)
CHPF$  template d(N,N)
CHPF$  distribute d(block,block) onto p
CHPF$  align WITH d :: A,B
```

```
      A=0
      K1=N-1
      K2=N/2

      IF(.NOT.DEBUG) THEN
C
      print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
C      start_timer
      CALL system_clock(tp,tr)

      do it=1,4
        FORALL(I=1:N,J=1:N) B(I,J)= I/(J+1.0)
        A(1:N:2,K1)=B(1:M,K2)
        FORALL(I=2:M) A(I,K1)=A(2*I-1,K1)*B(I,1)
      enddo

C         stop_timer
     CALL system_clock(ta)

      ELSE

        FORALL(I=1:N,J=1:N) B(I,J)= I/(J+1.0)

         write(6,*) ' ---->  B(I,J)=I/(J+1.0)'
        DO i=1,3
         write(6,*) B(i,5),' should be',i/6.0
        ENDDO
        DO i=N-2,N
         write(6,*) B(i,5),' should be',i/6.0
        ENDDO
        write(6,*) ' '

      A(1:N:2,K1)=B(1:M,K2)

         write(6,*) ' ---->  A(1:N:2,K1)=B(1:M,K2)'
         DO L=1,5
          ll=mod(l,2)
          write(6,*) a(l,k1),' should be', (l/2+1)/(k2+1.0)*ll
         ENDDO
         DO L=N-4,N
          ll=mod(l,2)
          write(6,*) a(l,k1),' should be', (l/2+1)/(k2+1.0)*ll
         ENDDO
         write(6,*) ' '

      FORALL(I=2:M) A(I,K1)=A(2*I-1,K1)*B(I,1)

         write(6,*) ' ----> FORALL(I=2:M) A(I,K1)=A(2*I-1,K1)*B(I,1)'
         DO L=2,5
         q=l/(k2+1.0)
         x1=l/2.0
         y1=q*x1
         write(6,*) a(l,k1),' should be',y1
        ENDDO
         DO L=M-3,M
         q=l/(k2+1.0)
         x1=l/2.0
         y1=q*x1
         write(6,*) a(l,k1),' should be',y1
        ENDDO
      ENDIF

     print *,' '
     print *,'******************************'
     print *,'         PARKBENCH '
     print *,' LOW LEVEL HPF BENCHMARK SUITE'
     print *,'         ST  KERNEL'
     print *,' PROBLEM SIZE: N =', N
     print *,' FINAL RESULT = ', A(M-1,K1)
```

```
      print *,' EXPECTED VALUE FOR SIZE 2048'
      print *,' IS  510.502 +/- 0.001'
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      STOP
      END
```

A.2.9 tm.hpf

```
      PROGRAM TM
C     ================================================================
C     template kerenel
C     adopted from the Purdue Set  (J. Rice)
C     ================================================================
C     please send comments to haupt@npac.syr.edu

      INTEGER NDIM,MDIM,ND1,MD1
C      variabili per il timer
      integer tp, ta, tr
      PARAMETER (NDIM=1023,MDIM=2047,ND1=NDIM+1,MD1=MDIM+1)
      REAL, DIMENSION(MDIM)          :: R
      REAL, DIMENSION(NDIM)          :: C
      REAL, DIMENSION(NDIM,MDIM)     :: A
      REAL, DIMENSION(ND1,MD1)       :: ABIG
      REAL, DIMENSION(1,1)           :: ACORN
      REAL, DIMENSION(NDIM-1,MDIM-1) :: B
      LOGICAL DEBUG
c      DATA DEBUG /.FALSE./
      DATA DEBUG /.TRUE./

CHPF$ PROCESSORS P(NUMBER_OF_PROCESSORS())
CHPF$ TEMPLATE TMPL(ND1)
CHPF$ DISTRIBUTE TMPL(BLOCK) onto P
CHPF$ ALIGN ABIG(i,*) with TMPL(i)
CHPF$ ALIGN A(i,*) with TMPL(i)
CHPF$ ALIGN C(:) with TMPL(:)
CHPF$ ALIGN B(i,*) with TMPL(i+2)

      print *,'Numero di processori:', NUMBER_OF_PROCESSORS()
c      start_timer
      CALL system_clock(tp,tr)

      do it=1,3
      forall(i=1:mdim) r(i)=1.0+i
      forall(i=1:ndim) c(i)=1.0-i
      forall(i=1:ndim,j=1:mdim) a(i,j)=i+j
      acorn = 0.5

      ABIG(1:NDIM,1:MDIM)=A
      ABIG(1:NDIM,MD1)=C
      ABIG(ND1,1:MDIM)=R
      ABIG(ND1,MD1)=ACORN(1,1)


      FORALL(I=1:ND1-2,J=1:MD1-2) B(I,J)=ABIG(I+2,J+2)

      enddo
c      stop_timer()
      CALL system_clock(ta)

      print *,' '
      print *,'******************************'
      print *,'         PARKBENCH '
      print *,' LOW LEVEL HPF BENCHMARK SUITE'
      print *,'         TM  KERNEL'
      print *,' PROBLEM SIZE: N, M =',ND1,MD1
```

```
      print *,' FINAL RESULT = ',B(ndim-2,mdim-2)
      print *,' EXPECTED VALUE FOR', ND1,' by',MD1
      print *,'       is', ndim+mdim
      print *,'******************************'

c     print execution time
      print *,'Tempo d''esecuzione:', (ta-tp)

      IF(DEBUG) THEN
        write(6,*) abig(nd1,md1),' and should be', acorn(1,1)
        write(6,*) abig(ndim,mdim),' and should be', ndim+mdim
        write(6,*) abig(ndim,md1),' and should be',1-ndim
        write(6,*) abig(nd1,mdim),' and should be',1+mdim
        write(6,*) b(ndim-1,mdim-1),' and should be', acorn(1,1)
      ENDIF

      STOP
      END
```

A.3 Purdue Set

A.3.1 Problem 01

```
      PROGRAM PROB01

C
C     PROBLEM 1
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      ************************************************
C      *      Adapted for FORTRAN D benchmarking      *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C      *                                              *
C      *    Northeast Parallel Architectures Center   *
C      *   at Syracuse University, Syracuse, NY, USA  *
C      ************************************************
C
C
C       VERSION HPF
C       ==================================================
C

      INTEGER KASES,K,NK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=6)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
      REAL TN
      REAL A,B,H
      DATA A,B /0.0,1.0/
      DATA N / 8192, 65536,
     X         131072, 262144, 524288, 1048576/

      DO K = 1, KASES
       NK=N(K)
        CALL SYSTEM_CLOCK (TSTART,TRATE)
        DO MANY=1,100

        H = (B-A)/(NK-1)
        CALL DOIT(NK-1,A,H,TN)
        TN = H*((EXP(A)+EXP(B))/2.+TN)

        ENDDO
```

```
        CALL SYSTEM_CLOCK(TSTOP)

        PRINT 30,A,B,NK
   30   FORMAT ('PROBLEM 1 WITH A,B,N = ',2F10.3,5X,I8)
      PRINT 40,TN
   40 FORMAT ('GIVES TN =',F10.6)

      PRINT *,'TIME = ',REAL(TSTOP-TSTART)/REAL(TRATE)

       ENDDO

       END



       SUBROUTINE DOIT(N,A,H,FSUM)
       INTEGER N
       REAL A,H,FSUM
       REAL R(N)

       R = EXP(A + H*[1:N])
       FSUM=SUM(R)

       END
```

A.3.2 Problem 02

```
      PROGRAM PROB02
C
C     PROBLEM 2
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ==================================================
C
      INTEGER KASES,NK,MK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N / 128,1024,1024,1024 /
      DATA M / 128,64,512,1024 /

      DO 50 K = 1, KASES

      CALL SYSTEM_CLOCK (TSTART,TRATE)

      DO MANY=1,25
         NK=N(K)
         MK=M(K)
         CALL DOIT(NK,MK,ESTAR)
      ENDDO

      CALL SYSTEM_CLOCK (TSTOP)
```

```
      PRINT 60, N(K),M(K)
   60 FORMAT ('PROBLEM 2 WITH N,M =',I6,2X,I6)
      PRINT *,'GIVES ESTAR =',ESTAR

      PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
      END


      SUBROUTINE DOIT(NK,MK,ESTAR)
      INTEGER NK,MK
      REAL ESTAR
      REAL, ARRAY(MK,NK) :: TEMP
      REAL, ARRAY(NK)    :: H

!HPF$ ALIGN H(i) WITH TEMP(*,i)
!HPF$ DISTRIBUTE TEMP (BLOCK,BLOCK)

      TEMP=1.+0.5/(REAL(-IABS(SPREAD([1:MK],2,NK)
     *                          -SPREAD([1:NK],1,MK)))+0.001)
      H = PRODUCT (TEMP, DIM=1)
      ESTAR=SUM(H)

      END
```

A.3.3 Problem 03

```
      PROGRAM PROB03
C
C     PROBLEM 3
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ==================================================
C
      INTEGER KASES,NK,MK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=3)
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N / 64,512,1024 /
      DATA M /128,128,1024 /

      REAL S


      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,50
```

```
          NK=N(K)
          MK=M(K)
          CALL DOIT(NK,MK,S)

        ENDDO

        CALL SYSTEM_CLOCK (TSTOP)

        PRINT 60, N(K),M(K)
   60   FORMAT ('PROBLEM 3 WITH N,M =',I6,2X,I6)
        PRINT *,'GIVES S =',S

        PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
      STOP
      END

      SUBROUTINE DOIT(NK,MK,S)
      INTEGER NK,MK
      REAL S

c     in original version the dimensions of A were interchanged
c
c     REAL, ARRAY (NK,MK) :: A
c     S=SUM(PRODUCT(A,DIM=2))

      REAL, ARRAY (MK,NK) :: A
      REAL, ARRAY (NK) :: H
!hpf$ align H(i) with A(*,i)

C
C      intitialization
C
      A=1.0001

C
C     computation
C

      H = PRODUCT (A, DIM=1)
      S = SUM (H)

      END
```

A.3.4 Problem 04

```
      PROGRAM PROB04
C
C     PROBLEM 04
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       *************************************************
C       *      Adapted for FORTRAN D benchmarking       *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)    *
C       *                                               *
C       *    Northeast Parallel Architectures Center    *
C       *   at Syracuse University, Syracuse, NY, USA   *
C       *************************************************
C
C
C        VERSION HPF
C        ===================================================
C
```

```
      INTEGER KASES,NK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=3)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE (*) :: N
      DATA N /16384,65536,524288/
      REAL X

      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,25

            NK=N(K)
            CALL DOIT(NK,X)

         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT *,'PROBLEM 4 WITH N =',NK
         PRINT *,'GIVES X  = ',X

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
      STOP
      END

      SUBROUTINE DOIT(NK,X)
      INTEGER NK
      REAL X
      REAL, ARRAY(NK)    :: Y,Z
!HPF$ ALIGN Z(I) WITH Y(I)

C     Y=1.0/(1.0+[1:NK])

      DO MANY=1,20
         Y = MANY
         WHERE (Y .ne. 0)
            Z = 1.0 / Y
          ELSEWHERE
            Z = 0.0
         ENDWHERE
      ENDDO

      X=SUM(Z)

      END
```

A.3.5 Problem 05

```
      PROGRAM PROB05
C
C     PROBLEM 5
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking       *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)    *
C       *                                               *
C       *    Northeast Parallel Architectures Center    *
```

```
C      *   at Syracuse University, Syracuse, NY, USA   *
C      ************************************************
C
C
C       VERSION HPF
C       ================================================
C
      INTEGER KASES,NS,NT
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=5)
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N / 64,1024,64,256,128 /
      DATA M / 128,64,1024,256,4092 /
      INTEGER NABOVE
      REAL AVER,AVERTOP,LOWABO
      LOGICAL GENIUS

      DO 50 K = 1, KASES

      CALL SYSTEM_CLOCK (TSTART, TRATE)

      DO MANY=1,200
         NS=N(K)
         NT=M(K)
         CALL DOIT(NS,NT,AVER,AVERTOP,GENIUS,NABOVE,LOWABO)
      ENDDO

      CALL SYSTEM_CLOCK (TSTOP)

      PRINT 60, NS,NT
   60 FORMAT ('PROBLEM 5 WITH ',I6,' STUDENTS AND ',I6,' TESTS')
      PRINT *,'AVERAGE TEST SCORE .....:',AVER
      PRINT *,'# SCORES ABOVE AVERAGE..:',NABOVE
      PRINT *,'AVERAGE ABOVE ..........:',AVERTOP
      PRINT *,'LOWEST SCORE ABOVE .....:',LOWABO
      PRINT *,'THERE IS GENIUS ........:',GENIUS

      PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE

c     STOP
      END

      SUBROUTINE DOIT(NS,NT,AVER,AVERTOP,GENIUS,NABOVE,LOWABO)
      INTEGER NABOVE
      REAL AVER,AVERTOP,LOWABO
      LOGICAL GENIUS
      REAL, ARRAY(NT,NS) :: SCORES
      LOGICAL, ARRAY(NT,NS) :: ABOVE

!HPF$ DISTRIBUTE SCORES (*,BLOCK)
!HPF$ ALIGN ABOVE(I,J) with SCORES(I,J)

         SCORES=60.0+40.0*SIN(SPREAD([1:NT],2,NS)*
     +                    SPREAD([1:NS],1,NT)*0.0006321)

         SSUM=SUM(SCORES)
         AVER=SSUM/(NS*NT)

         ABOVE = (SCORES.GT.AVER)

         WHERE(ABOVE)
           SCORES=SCORES*1.1
         ENDWHERE

         NABOVE=COUNT(ABOVE)
         AVERTOP=SUM(SCORES,MASK=ABOVE)/NABOVE
         LOWABO=MINVAL(SCORES,MASK=ABOVE)
```

```
         GENIUS=ANY(ALL(ABOVE,DIM=1))

      RETURN
      END
```

A.3.6 Problem 06

```
      PROGRAM PROB06
C
C     PROBLEM 6
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ================================================
C
      INTEGER KASES,K,NK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
      REAL SOLUT
      DATA N / 8196,16384,65536,262144/
C
C              LOOP OVER KASES
C
      DO  K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         NK=N(K)

         DO MANY=1,20
            CALL DOIT(NK,SOLUT)
         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT *,'PROBLEM 6 WITH N = ',NK
         PRINT *,'GIVES SOLUTION =', SOLUT

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

      ENDDO

      STOP
      END


      SUBROUTINE DOIT(NK,SOLUT)
      INTEGER NK
      REAL SOLUT
      DOUBLE PRECISION, ARRAY(NK) :: L,D,T,X,Y,U
!HPF$ ALIGN WITH L :: D, T, X, Y, U
```

```
      INTEGER II,K,LIMIT

c      L=0.88-0.1*SIN([1:NK]*12.36)
c      D=1.0d00+0.01*COS([1:NK]*8.11)
c      U=0.75+0.2*SIN([1:NK]*36.12+3.2)
      L=1.0d00
      D=0.5d00
      U=0.5d00
      Y=1.0d00
      X=0.0
      T=0.0
C
C
C      LIMIT = LOG BASE 2 OF N
C
        LIMIT = 1.44269504*ALOG(FLOAT(NK))+.01
        K = 1
C
C                              MAIN LOOP
C
        DO II = 1, LIMIT

              L=L/D
              U=U/D
              Y=Y/D

C
C                       T IS A TEMPORARY ARRAY
C                 COMPUTE AND ASSIGN TO D, COMPUTE Y
C
          D(1:K) = 1.0    - U(1:K)*L(K+1:K+K)
          T(1:K) = Y(1:K) - U(1:K)*L(K+1:K+K)

          D(K+1:NK-K) = 1.0 - L(K+1:NK-K)*U(1:NK-2*K) -
     +                 U(K+1:NK-K)*L(2*K+1:NK)
          T(K+1:NK-K) = Y(K+1:NK-K)   - L(K+1:NK-K)*Y(1:NK-2*K) -
     +                 U(K+1:NK-K)*L(2*K+1:NK)

          D(NK-K+1:NK) = 1.0         - L(NK-K+1:NK)*U(NK-2*K+1:NK-K)
          T(NK-K+1:NK) = Y(NK-K+1:NK) - L(NK-K+1:NK)*Y(NK-2*K+1:NK-K)
C
C                       ASSIGN TO Y, COMPUTE L
C
           Y=T
           T(1:K)=0
           T(K+1:NK)=-L(K+1:NK)*L(1:NK-K)
C
C                       ASSIGN TO L, COMPUTE U
C
           L=T
           T(1:NK-K)=U(1:NK-K)*U(K+1:NK)
           T(NK-K+1:NK)=0
C
C                       ASSIGN TO U
C
           U=T

           K = 2*K

      ENDDO

      X=Y/D
      SOLUT=SUM(X)

      RETURN
      END
```

A.3.7 Problem 07

```
      PROGRAM PROB07
C
C     PROBLEM 07
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      *************************************************
C      *      Adapted for FORTRAN D benchmarking       *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)    *
C      *                                               *
C      *    Northeast Parallel Architectures Center    *
C      *   at Syracuse University, Syracuse, NY, USA   *
C      *************************************************
C
C
C       VERSION HPF
C       ==================================================
C
      INTEGER KASES,NK,MK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=3)
      INTEGER N(KASES),M(KASES)
      DATA N /3,5,10/
      DATA M /2048,16384,32768/
      REAL P(4)

!HPF$ DISTRIBUTE (*) :: N, M, P

      DO K =  1, KASES

         NK=N(K)

         DO I = 1, KASES

            CALL SYSTEM_CLOCK (TSTART,TRATE)

            DO MANY=1,50
               MK=M(I)
               CALL DOIT(NK,MK,P)
            ENDDO
            CALL SYSTEM_CLOCK (TSTOP)


            PRINT *,'PROBLEM 7 WITH N,M =',NK,MK
            WRITE (6,*) 'GIVES P(',P(1),')  =',P(2)
            WRITE (6,*) 'AND   P(',P(3),')  =',P(4)

            PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

         END DO
      END DO
      STOP
      END

      SUBROUTINE DOIT(NK,MK,P)

c     note : distribution of 1:MK is only considered

      INTEGER NK,MK
      REAL P(4)
!hpf$ distribute (*) :: P

      REAL DX
      REAL, ARRAY(NK)       :: XI,DENOM,F
```

```
!hpf$ distribute (*)         :: XI,DENOM,F
      REAL, ARRAY(MK)        :: X,TP

      REAL, ARRAY(NK,MK)     :: TAMP,XL
      LOGICAL, ARRAY(NK,MK)  :: MASK1

!hpf$ align with TAMP       :: XL, MASK1
!hpf$ align (i) with TAMP(*,i) :: X, TP

      REAL, ARRAY(NK,NK)     :: TEMP
      LOGICAL, ARRAY(NK,NK)  :: MASK
      LOGICAL, ARRAY(NK)     :: MASK0

!hpf$ distribute (*,*) :: TEMP, MASK
!hpf$ distribute (*)   :: MASK0

      INTEGER I,K

      XI = [1:NK]-1.0
      DX=XI(NK)/MK

      X =  0.5+([1:MK]-1.0)*DX
      CALL FUN(XI,NK,F)

      MASK0 = .FALSE.

c     workaround for MASK = DIAGONAL(MASK0,FILL=.TRUE.)
      MASK  = .TRUE.
      FORALL (I=1:NK) MASK (I,I) = MASK0(I)

C     to jest macierz 3x3, 5x5, lub 10x10 TYLKO!

      TEMP =SPREAD(XI,2,NK)-SPREAD(XI,1,NK)
      DENOM=PRODUCT(TEMP,DIM=2,MASK=MASK)

      TAMP=SPREAD(X,1,NK)-SPREAD(XI,2,MK)
      DO I=1,NK
          MASK1=SPREAD(MASK(1:NK,I),2,MK)
          XL(I,1:MK)=PRODUCT(TAMP,DIM=1,MASK=MASK1)/DENOM(I)
      ENDDO

c     workaround for TP=SUM(SPREAD(F,2,MK)*XL,DIM=1)

      XL = SPREAD(F,2,MK)*XL
      TP=SUM(XL,DIM=1)

      P(1)=X(1)
      P(2)=TP(1)
      P(3)=X(MK)
      P(4)=TP(MK)

      RETURN
      END

      SUBROUTINE FUN(X,NK,F)
      INTEGER NK
      REAL, ARRAY(NK) :: X,F
!HPF$ distribute (*) :: X, F
      F = X**2-3.0*X-4.0
      RETURN
      END
```

A.3.8 Problem 08

```
      PROGRAM PROB08
C
C     PROBLEM 08
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
```

```
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      ************************************************
C      *     Adapted for FORTRAN D benchmarking       *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C      *                                              *
C      *    Northeast Parallel Architectures Center   *
C      *   at Syracuse University, Syracuse, NY, USA  *
C      ************************************************
C
C
C       VERSION HPF
C       ==================================================
C
      INTEGER KASES,NK,MK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N / 8192,65536,131072,65536/
      DATA M /3,4,4,8/
      REAL SUMD

      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,10
            NK=N(K)
            MK=M(K)
            CALL DOIT(NK,MK,SUMD)
         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT *,'PROBLEM 08 WITH N,M  =',NK,MK
         PRINT *,'GIVES SUM OF DIFFERENCES = ',SUMD

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
      STOP
      END

      SUBROUTINE DOIT(NK,MK,SUMD)
      INTEGER NK,MK
      REAL SUMD
      INTEGER J,N
      REAL, ARRAY(NK)    :: X, X1
      REAL, ARRAY(MK,NK) :: DIFF
      REAL, ARRAY(NK)    :: DIFF_ROW

!HPF$ DISTRIBUTE DIFF (*,BLOCK)
!HPF$ ALIGN (I) WITH DIFF(*,I) :: X

      DIFF = 0.0

      X=0.2*[1:NK]+0.01*COS(REAL([1:NK]))
      DIFF(1,1:NK)=SIN(X)

      X1 = X
      N=NK
      DO J=2,MK
        N=N-1
        DIFF(J,1:N)= (DIFF(J-1,2:N+1) - DIFF(J-1,1:N))
     &              / (X(J:N+J-1) - X(1:N))
```

```
      ENDDO

      SUMD=SUM(DIFF(2:MK,1:NK))

      END
```

A.3.9 Problem 09

```
      PROGRAM PROB09
C
C     PROBLEM 9
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ==================================================
C
      INTEGER KASES,NK,MK, NINT
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N /64,256,256,512/
      DATA M /128,128,256,512/

      PRINT *, 'PROBLEM 9 started'


      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,3
            NK=N(K)
            MK=M(K)
            CALL DOIT(NK,MK,TSUMT,NINT)
         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT 60, N(K),M(K)
  60     FORMAT ('PROBLEM 9 WITH N,M =',I6,2X,I6)
         PRINT *,'GIVES TSUMT =',TSUMT
         PRINT *,'AFTER ',NINT,' ITERATIONS'

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE

c     STOP
      END


      SUBROUTINE DOIT(NK,MK,TSUMT,NINT)
      INTEGER NK,MK
```

```
      INTEGER NINT
      REAL TSUMT,TOLER,ERROR
      DATA TOLER /0.05/
      REAL, ARRAY(NK,MK) :: U, T, ERRM
C
C     intitialization
C
      NINT=0
      ERROR=1.0E10

      FORALL(J=1:MK,I=1:NK) U(I,J) = I*(I+1)+FLOAT(J)/(J+1)
      T = 0.0
C
C      update U
C
       DO WHILE (ERROR.GT.TOLER)

       NINT=NINT+1

c      update inner board

       FORALL (J=2:MK-1,I=2:NK-1)
     $       T(I,J) = (U(I,J) + U(I+1,J) + U(I-1,J)
     $          + U(I,J+1) + U(I+1,J+1) + U(I-1,J+1)
     $          + U(I,J-1) + U(I+1,J-1) + U(I-1,J-1) ) / 9.

c      update left column

       FORALL (I=2:NK-1)
     $       T(I,1) = (U(I,1) + U(I+1,1) + U(I-1,1)
     $            + U(I,2) + U(I+1,2) + U(I-1,2) ) / 6.

c      update right column

       FORALL (I=2:NK-1)
     $       T(I,MK) = (U(I,MK) + U(I+1,MK) + U(I-1,MK)
     $            + U(I,MK-1) + U(I+1,MK-1) + U(I-1,MK-1) ) / 6.

c      update top row

       FORALL (J=2:MK-1)
     $    T(1,J) = (U(1,J) + U(2,J) + U(1,J+1)
     $        + U(2,J+1) + U(1,J-1) + U(2,J-1) ) / 6.

c      update bottom row

       FORALL (J=2:MK-1)
     $    T(NK,J) = (U(NK,J) + U(NK-1,J) + U(NK,J+1)
     $        + U(NK-1,J+1) + U(NK,J-1) + U(NK-1,J-1) ) / 6.

c      update corners
       T(1,1) = (U(1,1)+U(1,2)+U(2,2)+U(2,1)) / 4.
       T(1,MK) = (U(1,MK)+U(1,MK-1)+U(2,MK-1)+U(2,MK)) / 4.
       T(NK,1) = (U(NK,1)+U(NK,2)+U(NK-1,2)+U(NK-1,1)) / 4.
       T(NK,MK) = (U(NK,MK)+U(NK,MK-1)+U(NK-1,MK-1)+U(NK-1,MK)) / 4.

       WHERE (ABS(U).GT.0.001)
          ERRM=ABS((U-T)/U)
       ELSEWHERE
          ERRM=ABS((U-T)/0.001)
       ENDWHERE

       ERROR=MAXVAL(ERRM)

       U=T

c       ERROR=0
       END DO
```

```
      TSUMT= SUM(U(1:NK,1:MK))

      RETURN
      END
```

A.3.10 Problem 11

```
      PROGRAM PROB11
C
C     PROBLEM 11
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      *************************************************
C      *      Adapted for FORTRAN D benchmarking       *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)    *
C      *                                               *
C      *    Northeast Parallel Architectures Center    *
C      *   at Syracuse University, Syracuse, NY, USA   *
C      *************************************************
C
C
C       VERSION HPF
C       ==================================================
C
      INTEGER KASES,NK,MK,NFMOM
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      PARAMETER (NFMOM=4)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
      DATA N / 16384,32768,65536,262144/
      REAL TFMOM(NFMOM)
!HPF$ DISTRIBUTE TFMOM(*)

      DO K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,10
            NK=N(K)
            CALL DOIT(NK,TFMOM)
         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT *,'PROBLEM 11 WITH N =',NK

         DO I=1,NFMOM
            PRINT *,'MOMENT ',I,' = ',TFMOM(I)
         ENDDO

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

      END DO    ! different KASES

      STOP
      END

      SUBROUTINE DOIT(NK,TFMOM)
      INTEGER NK,NFMOM
      PARAMETER (NFMOM = 4)
      REAL TFMOM (NFMOM)
!HPF$ DISTRIBUTE(*) :: TFMOM
```

```
      REAL PI
      DATA PI / 3.1415926 /
      REAL, ARRAY(NK)     :: DATES,COSWT

      DATES=-[1:NK]*10.0 + 1080.0*SIN(1+[1:NK]*0.0623)
      DATES = AMAX1(0.0,AMIN1(1000.,DATES))
      DATES = ALOG(DATES+1.0)

      TFMOM(1) = SUM(DATES)/NK

      DO K = 2, NFMOM
        COSWT = COS(PI*[1:NK]*(K-1)/(NK+1)) * DATES
        TFMOM(K) = SUM(COSWT)/NK
      ENDDO


      RETURN
      END
```

A.3.11 Problem 12

```
      PROGRAM PROB12
C
C     PROBLEM 12
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ==================================================
C
      INTEGER KASES,NK,MK
      PARAMETER (KASES=4)
      INTEGER TSTART, TSTOP, TRATE
      INTEGER N(KASES),M(KASES)
!HPF$ DISTRIBUTE (*) :: N, M
      DATA N / 127,127,511,1023 /
      DATA M / 127,511,511,511 /
      REAL TP1,TP2

      DO K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

         DO MANY=1,150
            NK=N(K)
            MK=M(K)
            CALL DOIT(NK,MK,TP1,TP2)
         ENDDO

         CALL SYSTEM_CLOCK (TSTOP)

         PRINT 80,NK,MK
   80    FORMAT ('PROBLEM 12 WITH N,M =',I6,2X,I6)
         PRINT*,'GIVES CORNER PRODUCTS =', TP1,TP2
```

```
         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

      END DO   ! different kases

      END


      SUBROUTINE DOIT(NK,MK,TP1,TP2)
      INTEGER NK,MK
      REAL TP1,TP2
      REAL, ARRAY(NK+1,MK+1)    :: ABIG
      REAL ACORN


      ACORN = .5
      ABIG(1:NK,1:MK)=SPREAD([1:NK],2,MK)+SPREAD([1:MK],1,NK)
      ABIG(1:NK,MK+1:MK+1)=spread (1.0-[1:NK], 2, 1)
      ABIG(NK+1,1:MK)=1.0+[1:MK]

      ABIG(NK+1,MK+1)=ACORN


      TP1 = ABIG(1,1)*ABIG(1,MK+1)*ABIG(NK+1,MK)*ABIG(NK+1,MK+1)
      TP2 = ABIG(NK,MK)*ABIG(NK,MK+1)*ABIG(NK+1,MK)*ABIG(NK+1,MK+1)

      RETURN

      END
```

A.3.12 Problem 13

```
      PROGRAM PROB13
C
C     PROBLEM 13
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      ************************************************
C      *      Adapted for FORTRAN D benchmarking      *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C      *                                              *
C      *    Northeast Parallel Architectures Center   *
C      *   at Syracuse University, Syracuse, NY, USA  *
C      ************************************************
C
C
C       VERSION HPF
C       ==================================================
C
      INTEGER KASES,NK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
      DATA N / 8192, 16384,65536,262144 /
      REAL E

      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

      DO MANY=1,100
      NK=N(K)
      CALL DOIT(NK,E)
```

```
      ENDDO
         CALL SYSTEM_CLOCK (TSTOP)

      PRINT *,'PROBLEM 13 WITH N =',NK
      PRINT *,'GIVES E  = ', E


         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
c     STOP
      END

      SUBROUTINE DOIT(NK,E)
      INTEGER NK
      REAL E
      REAL, ARRAY(NK)    :: A,B,C,D
      LOGICAL, ARRAY (NK) :: MASK

         A = [1:NK]/10.0 + 1.0/[1:NK]
         B = LOG10(A) + 0.02
         C = (A + B) * SIN(A)
         D = A + B - 2.0*C

        A = A ** SIN(B)

        MASK = (SIN(A) .lt. COS(C))
        WHERE(MASK)
         A=A+C
        ELSEWHERE
         A=A-D
        END WHERE

        E=SUM(A**2)


c     RETURN
      END
```

A.3.13 Problem 14

```
      PROGRAM PROB14
C
C     PROBLEM 14
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C       ************************************************
C       *      Adapted for FORTRAN D benchmarking      *
C       *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C       *                                              *
C       *    Northeast Parallel Architectures Center   *
C       *   at Syracuse University, Syracuse, NY, USA  *
C       ************************************************
C
C
C        VERSION HPF
C        ================================================
C
      INTEGER KASES,JFUNK,NFUNK
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=4)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
```

```
      DATA N / 8192,16384,65536,262144/
      DATA NFUNK /3/
      INTEGER METH,IFUN,NK
      REAL RESULT,TRUE,A,B,ERROR

      DO  IFUN=1,NFUNK

         CALL FVALS(A,B,TRUE,IFUN)

         DO  K = 1, KASES

            NK=N(K)

            DO METH=1,3

               CALL SYSTEM_CLOCK (TSTART,TRATE)

               DO MANY=1,50
                  CALL DOIT(NK,A,B,METH,IFUN,RESULT)
               ENDDO

               CALL SYSTEM_CLOCK (TSTOP)

               ERROR = RESULT - TRUE
               PRINT *, ' '
               PRINT *,'PROBLEM 14 WITH N = ',NK
               PRINT *,'METHOD',METH,' FUNCTION ',IFUN
               PRINT *,'GIVES INTEGRAL ESTIMATE =', RESULT
               PRINT *,'ERROR (ESTIMATE-TRUE VALUE) = ',ERROR
               PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

            ENDDO   ! METH
         ENDDO   ! KASES
      ENDDO   ! FUN
      END


      SUBROUTINE  DOIT(NK,A,B,METH,IFUN,RESULT)
      INTEGER NK,METH,IFUN
      REAL A,B,RESULT
      INTEGER NSIMP,NG
      REAL H77
      REAL, ARRAY(:)        :: X1,X2,X3,X
      REAL, ARRAY(:)        :: F1,F2,F3,F
      REAL H

C
      IF(METH.EQ.1) THEN
C
C        TRAPEZOIDAL RULE
C
           H = (B-A)/NK
           RESULT = 0

           allocate (X(0:NK), F(0:NK))
           X = A + H * [0:NK]
           CALL FUN(X,NK,IFUN,F)
           RESULT = (SUM(F(1:NK-1))*2.0+F(0)+F(NK))*H/2.0
           deallocate (F, X)

      ENDIF


      IF(METH.EQ.2) THEN
C
C        SIMPSON's METHOD
C
           NSIMP = NK
           IF (MOD(NSIMP,2).EQ.1) NSIMP = NSIMP-1
           H = (B-A)/NSIMP
```

```
           ALLOCATE (X(0:NSIMP), F(0:NSIMP))
           X = A + H * [0:NSIMP]
           CALL FUN(X,NSIMP,IFUN,F)
           RESULT=H*(F(0)+F(NSIMP)+4.0*SUM(F(1:NSIMP-1:2))+
     *           2.0*SUM(F(2:NSIMP-2:2)))/3.0
           DEALLOCATE (F, X)
      ENDIF

      IF(METH.EQ.3) THEN
C
C        GAUSS' METHOD
C
            NG=(NK-MOD(NK,3))/3
            H = (B-A)/NG
            H77 = .774596669241*H

            allocate (X1(0:NG), X2(0:NG), X3(0:NG))
            allocate (F1(0:NG), F2(0:NG), F3(0:NG))

            X1(0:NG)=A+H*[0:NG]-H/2.0-H77
            X2(0:NG)=A+H*[0:NG]-H/2.0
            X3(0:NG)=A+H*[0:NG]-H/2.0+H77


           CALL FUN(X1,NG,IFUN,F1)
           CALL FUN(X2,NG,IFUN,F2)
           CALL FUN(X3,NG,IFUN,F3)

c          CALL FUN(A+H*[0:NG]-H/2.0-H77,NG,IFUN,F1)
c          CALL FUN(A+H*[0:NG]-H/2.0,NG,IFUN,F2)
c          CALL FUN(A+H*[0:NG]-H/2.0+H77,NG,IFUN,F3)

           RESULT = H*(5.0*(SUM(F1(1:NG))+SUM(F3(1:NG)))+
     *             8.0*SUM(F2(1:NG)))/18.0
           DEALLOCATE (F3, F2, F1, X3, X2, X1)


      ENDIF

      END


      SUBROUTINE FUN(X,N,IFUN,F)
      INTEGER N,IFUN
      REAL X(0:N),F(0:N)

      IF (IFUN.EQ.1) F = EXP(X)
      IF (IFUN.EQ.2) F = SQRT(ABS(X-.2345))
      IF (IFUN.EQ.3) F = 1.+X*X+1./(1.+100.*X*X)
      END

      SUBROUTINE FVALS (A,B,TRUE,IFUN)
      IF (IFUN.EQ.1) THEN
         A = 0.
         B = 1.
         TRUE = 1.71828182845
      ENDIF
      IF (IFUN.EQ.2) THEN
         A = 0.
         B = 1.
         TRUE = .5222099422093
      ENDIF
      IF (IFUN.EQ.3) THEN
         A = -1.
         B = 2.
         TRUE = 6.29919656054
      ENDIF
      END
```

A.3.14 Problem 15

```
      PROGRAM PROB15
C
C     PROBLEM 15
C
C  REFERENCE:  PROBLEMS TO TEST PARALLEL AND VECTOR LANGUAGES
C              CSD-TR 516, COMPUTER SCIENCE, PURDUE UNIVERSITY
C              JOHN R. RICE, MAY 1, 1985
C
C              REVISED BY JOHN R. RICE AND J. JING, OCT. 1, 1990
C
C
C      ************************************************
C      *      Adapted for FORTRAN D benchmarking      *
C      *    by  T. HAUPT  (haupt@sccs.npac.syr.edu)   *
C      *                                              *
C      *    Northeast Parallel Architectures Center   *
C      *   at Syracuse University, Syracuse, NY, USA  *
C      ************************************************
C
C
C       VERSION HPF
C       ==================================================
C
      INTEGER KASES
      INTEGER TSTART, TSTOP, TRATE
      PARAMETER (KASES=2)
      INTEGER N(KASES)
!HPF$ DISTRIBUTE N(*)
      DATA N /8192,16384/
      INTEGER NK,NPTS
      PARAMETER(NPTS=10)
      REAL ERRMAX(NPTS,2),DECAY(NPTS,2)
!HPF$ DISTRIBUTE ERRMAX (*,*)
!HPF$ DISTRIBUTE DECAY (*,*)

      PRINT *, 'PROBLEM 15 started'
      DO 50 K = 1, KASES

         CALL SYSTEM_CLOCK (TSTART,TRATE)

      DO MANY=1,10
      NK=N(K)
      CALL DOIT(NK,ERRMAX,DECAY)
      ENDDO
         CALL SYSTEM_CLOCK (TSTOP)

      DO 170 KASE=1,2
         IF (KASE.EQ.1) THEN
            PRINT 110,NK
  110       FORMAT (4X,'PROBLEM 15 WITH NK=',I8/)
            PRINT 120
  120       FORMAT (9X,'EQUALLY SPACED POINTS'/)
            PRINT 130
  130       FORMAT (4X,'N      MAX ERROR      DECAY EXPONENT')
         ELSE
            PRINT 140
  140       FORMAT (/9X,'CHEBYSHEV SPACED POINTS'/)
            PRINT 130
         ENDIF
         DO 160 I = 2, NPTS
            PRINT 150,I,ERRMAX(I,KASE),DECAY(I,KASE)
  150       FORMAT (1X,I4,4X,E12.4,3X,F11.2)
  160    CONTINUE
C
C              PRINT OUT OF INTERPOLATION POINT ARRAY XPTS(J,N,KASE)
C              SUPPRESSED HERE TO REDUCE AMOUNT OF OUTPUT
```

```
C
  170 CONTINUE

         PRINT *, 'TIME = ', REAL(TSTOP-TSTART)/REAL(TRATE)

   50 CONTINUE
      STOP
      END

      SUBROUTINE DOIT(NK,ERRMAX,DECAY)
      INTEGER NK,NPTS
      PARAMETER(NPTS=10)
      REAL ERRMAX(NPTS,2),DECAY(NPTS,2)
!HPF$ DISTRIBUTE ERRMAX (*,*)
!HPF$ DISTRIBUTE DECAY (*,*)
      INTEGER NMAX,I,K,KNOTSMAX
      PARAMETER (NMAX=2*NPTS,KNOTSMAX=NPTS+2)
      REAL XPTS(NPTS),COEF(NMAX),H
!HPF$ DISTRIBUTE XPTS(*)
!HPF$ DISTRIBUTE COEF(*)
      REAL TKNOTS(KNOTSMAX)
!HPF$ DISTRIBUTE TKNOTS(*)
      REAL LOGERR,OLDERR,INTERP,RATIO,A,B
      DATA A,B / -1.,1. /,PI / 3.141592654 /
      INTEGER ERRTYP1,ERRTYP2
      DATA ERRTYP1,ERRTYP2 /31,32/
      REAL ERR1

      DO KASE = 1, 2
C
C        BEGIN             'NUMBER OF POINTS USED'
C
         DO 80 N = 2, NPTS

C
C           BEGIN                 'CASES OF POINT DISTRIBUTION'
C
            IF (KASE.EQ.1) THEN
               H = (B-A)/(N-1)
               DO 10 J = 1, N
                  XPTS(J) = A+(J-1)*H
   10          CONTINUE
            ELSE
               DO 20 J = 1, N
                  XPTS(J) = (A+B)/2+(A-B)/2*COS((2*J-1)*PI/(2*N))
   20          CONTINUE
               RATIO = (B-A)/(XPTS(N)-XPTS(1))
               DO 30 J = 1, N
                  XPTS(J) = (XPTS(J)-(A+B)/2)*
     *                                RATIO+(A+B)/2
   30          CONTINUE
            ENDIF

            DO 40 J = 1, N
               call F(XPTS(J), COEF(2*J-1))
   40       CONTINUE
            DO 50 J = 1, N
               call FPRIME (XPTS(J), COEF(2*J))
   50       CONTINUE

            DO 60 J = 1, N
               TKNOTS(J+1) = XPTS(J)
   60       CONTINUE
C
C           ADD THE DUMMY POINTS AT EACH END OF TKNOTS ARRAY
C
            TKNOTS(1) = A-0.1*(ABS(A)+1.)
            TKNOTS(N+2) = B+0.1*(ABS(B)+1.)
            KNOTS = N+2
```

```
            CALL INTERPOLATE(N,NK,A,B,COEF,TKNOTS,ERRMAX(N,KASE))

   80    CONTINUE

         DO 90 N = 3, NPTS
            OLDERR = ALOG(ERRMAX(N-1,KASE))
            LOGERR = ALOG(ERRMAX(N,KASE))
            DECAY(N,KASE) = (LOGERR-OLDERR)/ALOG(FLOAT(N)/FLOAT(N-1))
   90    CONTINUE

      ENDDO

c     RETURN
      END

      SUBROUTINE F (X,R)
      REAL X, R
      R = 1./(X*X+25.0)
      END

      SUBROUTINE FPRIME (X,R)
      REAL X, R
      R = -2.0*X/(X*X+25.)**2
      END

      SUBROUTINE FUN(X,NK,F)
      INTEGER NK
      REAL, ARRAY(NK) :: X,F
      F=1./(X*X+25.0)
      END

      SUBROUTINE INTERPOLATE(N,NK,A,B,COEF,T,ERROR)
      INTEGER NPTS,NMAX,KNOTSMAX
      PARAMETER (NPTS=10)
      PARAMETER (NMAX=2*NPTS,KNOTSMAX=NPTS+2)
      REAL COEF(NMAX),T(KNOTSMAX)
!HPF$ DISTRIBUTE COEF(*)
!HPF$ DISTRIBUTE T(*)
      REAL ERROR,A,B
      INTEGER N,NK,J,IKNOT
      REAL, ARRAY(NK) :: X,F,INTERP,DTDX,DTDX2,DX,COEF2
      REAL, ARRAY(NMAX,NK) :: HCUBIC

!HPF$ DISTRIBUTE HCUBIC (*,BLOCK)
!HPF$ ALIGN (I) WITH HCUBIC(*,I) :: X,F,INTERP,DTDX,DTDX2,DX,COEF2

      X=A+[0:NK-1]*(B-A)/(NK-1)
      CALL FUN(X,NK,F)
      COEF2=0

      DO J=1,2*N
       DTDX=0
       DTDX2=0
       DX=0

       IKNOT=(J+3)/2
       IF(MOD(J,2).EQ.1) THEN

         WHERE((X.LT.T(IKNOT+1)) .AND. (X.GT.T(IKNOT)))
             DTDX=(T(IKNOT+1)-X)/(T(IKNOT+1)-T(IKNOT))
         ENDWHERE

         WHERE((X.LE.T(IKNOT)) .AND. (X.GT.T(IKNOT-1)))
             DTDX=(X-T(IKNOT-1))/(T(IKNOT)-T(IKNOT-1))
         ENDWHERE

         WHERE((X.LT.T(IKNOT+1)) .AND. (X.GT.T(IKNOT-1)))
             HCUBIC(J,1:NK)=(3.0-2.0*DTDX(1:NK))*DTDX(1:NK)**2
```

```
          ELSEWHERE
            HCUBIC(J,1:NK)=0
          ENDWHERE

       ELSE

          WHERE((X.LT.T(IKNOT+1)) .AND. (X.GT.T(IKNOT)))
            DX=X-T(IKNOT)
            DTDX2=((X-T(IKNOT+1))/(T(IKNOT)-T(IKNOT+1)))**2
          ENDWHERE

          WHERE((X.LE.T(IKNOT)) .AND. (X.GT.T(IKNOT-1)))
            DX=X-T(IKNOT)
            DTDX2=((X-T(IKNOT-1))/(T(IKNOT)-T(IKNOT-1)))**2

          ENDWHERE

          WHERE((X.LT.T(IKNOT+1)) .AND. (X.GT.T(IKNOT-1)))
            HCUBIC(J,1:NK)=DTDX2(1:NK)*DX(1:NK)
          ELSEWHERE
            HCUBIC(J,1:NK)=0
          ENDWHERE

        ENDIF
        COEF2(1:NK)=COEF(J)*HCUBIC(J,1:NK)+COEF2(1:NK)
       ENDDO

C       INTERP=SUM(COEF2,DIM=1)
       INTERP=COEF2
       ERROR=MAXVAL(ABS(F-INTERP))

       END
```

A.4 Codice intermedio

A.4.1 Codice intermedio di Primi_Passi compilato con *ADAPTOR*

```
      PROGRAM PROVA
      INTEGER*4 N
      PARAMETER ( N = 100000 )
      INTEGER*4 SP_TOPID
      INTEGER*4 D_DSP
      REAL*4 D (1:2)
      INTEGER*4 D_DIM1
      INTEGER*4 D_ZERO
      INTEGER*4 C_DSP
      REAL*4 C (1:2)
      INTEGER*4 C_DIM1
      INTEGER*4 C_ZERO
      INTEGER*4 B_DSP
      REAL*4 B (1:2)
      INTEGER*4 B_DIM1
      INTEGER*4 B_ZERO
      INTEGER*4 A_DSP
      REAL*4 A (1:2)
      INTEGER*4 A_DIM1
      INTEGER*4 A_ZERO
      INTEGER*4 I
      INTEGER*4 TP
      INTEGER*4 TA
      INTEGER*4 TR
      INTEGER*4 I1
      INTEGER*4 ISEC_DSP2
      INTEGER*4 C_STOP1
      INTEGER*4 C_START1
      INTEGER*4 B_STOP1
      INTEGER*4 B_START1
      INTEGER*4 A_STOP1
      INTEGER*4 A_START1
```

```
EXTERNAL dalib_pid
INTEGER*4 dalib_pid
INTEGER*4 dalib_0
COMMON /dalib_data0/ dalib_0
EXTERNAL dalib_nproc
INTEGER*4 dalib_nproc
call dalib_init (4,4,4)
call dalib_set_present (dalib_0)
call dalib_start_subroutine ('PROVA',5)
call dalib_top_create (SP_TOPID,1,1,dalib_nproc())
call dalib_array_make_dsp (D_DSP,1,4)
call dalib_distribute (D_DSP,SP_TOPID,1,0)
call dalib_array_dynamic (D_DSP)
call dalib_array_define (D_DSP,1,N)
call dalib_array_allocate (D_DSP,D,D_ZERO,D_DIM1)
call dalib_array_make_dsp (C_DSP,1,4)
call dalib_distribute (C_DSP,SP_TOPID,1,0)
call dalib_array_dynamic (C_DSP)
call dalib_array_define (C_DSP,1,N)
call dalib_array_allocate (C_DSP,C,C_ZERO,C_DIM1)
call dalib_array_make_dsp (B_DSP,1,4)
call dalib_distribute (B_DSP,SP_TOPID,1,0)
call dalib_array_dynamic (B_DSP)
call dalib_array_define (B_DSP,1,N)
call dalib_array_allocate (B_DSP,B,B_ZERO,B_DIM1)
call dalib_array_make_dsp (A_DSP,1,4)
call dalib_distribute (A_DSP,SP_TOPID,1,0)
call dalib_array_dynamic (A_DSP)
call dalib_array_define (A_DSP,1,N)
call dalib_array_allocate (A_DSP,A,A_ZERO,A_DIM1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Numero di processori: ',dalib_nproc()
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_lslice (A_DSP,1,1,N,A_START1,A_STOP1)
DO I=A_START1,A_STOP1
   A(A_ZERO+I) = REAL(I)
END DO
call dalib_array_lslice (B_DSP,1,1,N,B_START1,B_STOP1)
DO I=B_START1,B_STOP1
   B(B_ZERO+I) = REAL(I)
END DO
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = REAL(I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Prima fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = A(A_ZERO+I)+B(B_ZERO+I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Seconda fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
```

```
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = A(A_ZERO+I)+B(B_ZERO+I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Terza fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I1=C_START1,C_STOP1
   C(C_ZERO+I1) = A(A_ZERO+I1)+B(B_ZERO+I1)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Quarta fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_section_create (ISEC_DSP2,A_DSP,1,N-1+1,N-N+1,-1)
call dalib_assign (C_DSP,ISEC_DSP2)
call dalib_section_free (ISEC_DSP2)
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = C(C_ZERO+I)+B(B_ZERO+I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Quinta fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_section_create (ISEC_DSP2,A_DSP,1,N-1+1,N-N+1,-1)
call dalib_assign (D_DSP,ISEC_DSP2)
call dalib_section_free (ISEC_DSP2)
call dalib_array_lslice (C_DSP,1,1,N,C_START1,C_STOP1)
DO I=C_START1,C_STOP1
   C(C_ZERO+I) = D(D_ZERO+I)+B(B_ZERO+I)
END DO
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Sesta fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_create_copy (A_NDSP,A_DSP)
call dalib_distribute (A_DSP,1,2,0)
call dalib_redistribute (A_DSP,A_NDSP)
call dalib_array_access (A_DSP,A,A_ZERO,A_DIM1)
call dalib_array_create_copy (B_NDSP,B_DSP)
call dalib_distribute (B_DSP,1,2,0)
```

```
call dalib_redistribute (B_DSP,B_NDSP)
call dalib_array_access (B_DSP,B,B_ZERO,B_DIM1)
call dalib_array_create_copy (C_NDSP,C_DSP)
call dalib_distribute (C_DSP,1,2,0)
call dalib_redistribute (C_DSP,C_NDSP)
call dalib_array_access (C_DSP,C,C_ZERO,C_DIM1)
call dalib_array_create_copy (D_NDSP,D_DSP)
call dalib_distribute (D_DSP,1,2,0)
call dalib_redistribute (D_DSP,D_NDSP)
call dalib_array_access (D_DSP,D,D_ZERO,D_DIM1)
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Settima fase: ',TA-TP
   call dalib_SYSTEM_CLOCK (TP,TR,dalib_0)
END IF
call dalib_broadcast (TP,4,1)
call dalib_broadcast (TR,4,1)
call dalib_array_create_copy (D_NDSP,D_DSP)
call dalib_distribute (D_DSP,1,1,0)
call dalib_redistribute (D_DSP,D_NDSP)
call dalib_array_access (D_DSP,D,D_ZERO,D_DIM1)
IF (dalib_pid() .eq. 1) THEN
   call dalib_SYSTEM_CLOCK (TA,dalib_0,dalib_0)
END IF
call dalib_broadcast (TA,4,1)
IF (dalib_pid() .eq. 1) THEN
   PRINT *,'Ottava fase: ',TA-TP
END IF
call dalib_array_free (A_DSP)
call dalib_array_free (B_DSP)
call dalib_array_free (C_DSP)
call dalib_array_free (D_DSP)
call dalib_end_subroutine ()
call dalib_exit ()
END
```

A.4.2 Codice intermedio di Primi_Passi compilato con *pghpf*

```
 program prova
 integer n
 parameter (n=100000)
 common /pghpf_0/pghpf_0, /pghpf_1/pghpf_1, /pghpf_0c/pghpf_0c
+, /pghpf_lineno/pghpf_lineno, /pghpf_01/pghpf_01
 integer pghpf_0, pghpf_1, pghpf_lineno, pghpf_01(8)
 character*1 pghpf_0c
 common /prova$a$dyn/ a$dp,a$d$o,a$p,a$o,l$b5,u$b5
 common /prova$b$dyn/ b$dp,b$d$o,b$p,b$o,l$b6,u$b6
 common /prova$c$dyn/ c$dp,c$d$o,c$p,c$o,l$b7,u$b7
 common /prova$d$dyn/ d$dp,d$d$o,d$p,d$o,l$b8,u$b8
 common /prova$c$f1$dyn/ c$f1$dp,c$f1$d$o,c$f1$p,c$f1$o,l$b,u$b
 common /prova$c$f3$dyn/ c$f3$dp,c$f3$d$o,c$f3$p,c$f3$o,l$b1,u$b1
 common /prova$c$f5$dyn/ c$f5$dp,c$f5$d$o,c$f5$p,c$f5$o,l$b2,u$b2
 common /prova$c$f7$dyn/ c$f7$dp,c$f7$d$o,c$f7$p,c$f7$o,l$b3,u$b3
 common /prova$c$f9$dyn/ c$f9$dp,c$f9$d$o,c$f9$p,c$f9$o,l$b4,u$b4
 integer hpf_np$, z_e_12, i, tp, ta, tr, z__io, a$dynamic,
+b$dynamic, c$dynamic, d$dynamic, a$tl$dynamic, b$tl$dynamic,
+c$tl$dynamic, d$tl$dynamic, ndi$vt, ndi$vt1, i$i, i$i1, i$i2, i$i3
+, i$i4, i$i5, i$i6, sp$d(1), c$lo, c$lo1, c$lo2, c$lo3, c$lo4,
+c$lo5, c$lo6, a$s(1), c$lo7, a$s1(1), c$lo8, c$l, c$u, c$s, c$lo9
+, c$ls, i$c, c$l1, c$u1, c$s1, c$lo10, c$ls1, i$c1, c$l2, c$u2,
+c$s2, c$lo11, c$ls2, i$c2, c$l3, c$u3, c$s3, c$lo12, c$ls3, i$c3,
+c$l4, c$u4, c$s4, c$lo13, c$ls4, i$c4, c$l5, c$u5, c$s5, c$lo14,
+c$ls5, i$c5, c$l6, c$u6, c$s6, c$lo15, c$ls6, i$c6, c$l7, c$u7,
+c$s7, c$lo16, c$ls7, i$c7, c$l8, c$u8, c$s8, c$lo17, c$ls8, i$c8,
+i$c9, i$l, i$u, i$c10, i$l1, i$u1, i$c11, i$l2, i$u2, i$c12, i$l3
+, i$u3, i$c13, i$l4, i$u4, i$c14, i$l5, i$u5, i$c15, i$l6, i$u6,
+i$c16, i$l7, i$u7, i$c17, i$l8, i$u8, a$d(1), b$d(1), c$d(1), d$d
```

```
+(1), a$tl$d(1), b$tl$d(1), c$tl$d(1), d$tl$d(1), a$tl$tl$d(1),
+b$tl$tl$d(1), c$tl$tl$d(1), d$tl$tl$d(1), c$f1$d(1), c$f3$d(1),
+c$f5$d(1), c$f7$d(1), c$f9$d(1), l$b, u$b, l$b1, u$b1, l$b2, u$b2
+, l$b3, u$b3, l$b4, u$b4, c$tl$tl$tl4$d(1), c$tl$tl$tl3$d(1),
+c$tl$tl$tl2$d(1), c$tl$tl$tl1$d(1), c$tl$tl$tl$d(1), l$b5, u$b5,
+l$b6
 integer u$b6, l$b7, u$b7, l$b8, u$b8
 real a(1), b(1), c(1), d(1), c$f(1), c$f1(1), c$f2(1), c$f3(1),
+c$f4(1), c$f5(1), c$f6(1), c$f7(1), d$f(1), c$f8(1), c$f9(1)
 integer cp, xfer, cp1, xfer1, cp2, xfer2, cp3, xfer3, cp4, xfer4,
+cp5, xfer5, cp6, xfer6, cp7, xfer7, cp8, xfer8, cp9, xfer9, cp10,
+xfer10
 integer a$p, a$o, b$p, b$o, c$p, c$o, d$p, d$o, sp$dp, c$f$p,
+c$f1$p, c$f1$o, c$f2$p, c$f3$p, c$f3$o, c$f4$p, c$f5$p, c$f5$o,
+c$f6$p, c$f7$p, c$f7$o, a$sp, d$f$p, a$sp1, c$f8$p, c$f9$p, c$f9$o
+, a$dp, a$d$o, b$dp, b$d$o, c$dp, c$d$o, d$dp, d$d$o, a$tl$dp,
+a$tl$d$o, b$tl$dp, b$tl$d$o, c$tl$dp, c$tl$d$o, d$tl$dp, d$tl$d$o
+, a$tl$tl$dp, a$tl$tl$d$o, b$tl$tl$dp, b$tl$tl$d$o, c$tl$tl$dp,
+c$tl$tl$d$o, d$tl$tl$dp, d$tl$tl$d$o, c$f1$dp, c$f1$d$o, c$f3$dp,
+c$f3$d$o, c$f5$dp, c$f5$d$o, c$f7$dp, c$f7$d$o, c$f9$dp, c$f9$d$o
+, c$tl$tl$tl4$dp, c$tl$tl$tl3$dp, c$tl$tl$tl2$dp, c$tl$tl$tl1$dp,
+c$tl$tl$tl$dp
 integer pgf90io_ldw_init, pgf90io_ldw, pgf90io_ldw_end,
+pghpf_nprocs, pghpf_comm_copy, pghpf_comm_start, pghpf_sect,
+pghpf_newproc
 data pghpf_01 /828006472,2089468329,886093537,-459164165,1,
+1261038733,0,-2056632774/
 call pghpf_init(0)
 hpf_np$ = pghpf_nprocs()
 z_e_12 = hpf_np$
 sp$dp = pghpf_newproc(1,hpf_np$)
 call pghpf_ptr_offset(sp$dp,sp$dp,sp$d,25)
 call pghpf_template(a$dp,1,34048,sp$d(sp$dp),0,0,1,100000)
 call pghpf_ptr_offset(a$d$o,a$dp,a$d,25)
 call pghpf_instance(a$dp,a$d(a$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(a$d$o,a$dp,a$d,25)
 call pghpf_template(b$dp,1,34048,sp$d(sp$dp),0,0,1,100000)
 call pghpf_ptr_offset(b$d$o,b$dp,b$d,25)
 call pghpf_instance(b$dp,b$d(b$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(b$d$o,b$dp,b$d,25)
 call pghpf_template(c$dp,1,34048,sp$d(sp$dp),0,0,1,100000)
 call pghpf_ptr_offset(c$d$o,c$dp,c$d,25)
 call pghpf_instance(c$dp,c$d(c$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(c$d$o,c$dp,c$d,25)
 call pghpf_template(d$dp,1,34048,sp$d(sp$dp),0,0,1,100000)
 call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
 call pghpf_instance(d$dp,d$d(d$d$o),27,4,0,0,0)
 call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
 call pghpf_allobnds(a$d(a$d$o),l$b5,u$b5)
 call pgf90_alloc(u$b5-l$b5+1,27,pghpf_0,a$p,a$o,a)
 call pghpf_allobnds(b$d(b$d$o),l$b6,u$b6)
 call pgf90_alloc(u$b6-l$b6+1,27,pghpf_0,b$p,b$o,b)
 call pghpf_allobnds(c$d(c$d$o),l$b7,u$b7)
 call pgf90_alloc(u$b7-l$b7+1,27,pghpf_0,c$p,c$o,c)
 call pghpf_allobnds(d$d(d$d$o),l$b8,u$b8)
 call pgf90_alloc(u$b8-l$b8+1,27,pghpf_0,d$p,d$o,d)
 call pgf90io_src_info(10,'prova.hpf')
 z__io = pgf90io_ldw_init(6,0,0,0)
 z__io = pgf90io_ldw(14,1,0,'Numero di processori: ')
 z__io = pgf90io_ldw(25,1,0,hpf_np$)
 z__io = pgf90io_ldw_end()
 call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!      call system_clock(tp,tr)
 continue
!      forall (i=1:100000)
 call pghpf_cyclic_loop(a$d(a$d$o),1,1,100000,1,c$l,c$u,c$s,c$lo9,
+c$ls)
!      forall (i=i$l:i$u) a(i-c$lo-l$b5+a$o) = i
 c$lo = c$lo9
 do i$c9 = c$l, c$u, c$s
```

```
            call pghpf_block_loop(a$d(a$d$o),1,1,100000,1,i$c9,i$l,i$u)
            do i = i$l, i$u
               a(i-c$lo-l$b5+a$o) = i
            enddo
            c$lo = c$lo + c$ls
         enddo
!        a(i-c$lo-l$b5+a$o) = i
         call pghpf_cyclic_loop(b$d(b$d$o),1,1,100000,1,c$l1,c$u1,c$s1,
     +c$lo10,c$ls1)
!        forall (i=i$l1:i$u1) b(i-c$lo1-l$b6+b$o) = i
         c$lo1 = c$lo10
         do i$c10 = c$l1, c$u1, c$s1
            call pghpf_block_loop(b$d(b$d$o),1,1,100000,1,i$c10,i$l1,i$u1)
            do i = i$l1, i$u1
               b(i-c$lo1-l$b6+b$o) = i
            enddo
            c$lo1 = c$lo1 + c$ls1
         enddo
!        b(i-c$lo1-l$b6+b$o) = i
         call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l2,c$u2,c$s2,
     +c$lo11,c$ls2)
!        forall (i=i$l2:i$u2) c(i-c$lo2-l$b7+c$o) = i
         c$lo2 = c$lo11
         do i$c11 = c$l2, c$u2, c$s2
            call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c11,i$l2,i$u2)
            do i = i$l2, i$u2
               c(i-c$lo2-l$b7+c$o) = i
            enddo
            c$lo2 = c$lo2 + c$ls2
         enddo
!        c(i-c$lo2-l$b7+c$o) = i
!        endforall
         call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!        call system_clock(ta)
         call pgf90io_src_info(20,'prova.hpf')
         z__io = pgf90io_ldw_init(6,0,0,0)
         z__io = pgf90io_ldw(14,1,0,'Prima fase: ')
         z__io = pgf90io_ldw(25,1,0,ta - tp)
         z__io = pgf90io_ldw_end()
         call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!        call system_clock(tp,tr)
         continue
!        forall (i=1:100000)
         cp = pghpf_comm_copy(c(c$o),a(a$o),c$d(c$d$o),a$d(a$d$o))
         xfer = pghpf_comm_start(cp,c(c$o),c$d(c$d$o),a(a$o),a$d(a$d$o))
         call pghpf_comm_finish(xfer)
         call pghpf_template(c$f1$dp,1,53248,c$d(c$d$o),0,1,100000)
         call pghpf_ptr_offset(c$f1$d$o,c$f1$dp,c$f1$d,25)
         call pghpf_instance(c$f1$dp,c$f1$d(c$f1$d$o),27,4,0,0,0)
         call pghpf_ptr_offset(c$f1$d$o,c$f1$dp,c$f1$d,25)
         call pghpf_allobnds(c$f1$d(c$f1$d$o),l$b,u$b)
         call pgf90_alloc(u$b-l$b+1,27,pghpf_0,c$f1$p,c$f1$o,c$f1)
         cp1 = pghpf_comm_copy(c$f1(c$f1$o),b(b$o),c$f1$d(c$f1$d$o),b$d(
     +b$d$o))
         xfer1 = pghpf_comm_start(cp1,c$f1(c$f1$o),c$f1$d(c$f1$d$o),b(b$o),
     +b$d(b$d$o))
         call pghpf_comm_finish(xfer1)
         call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l3,c$u3,c$s3,
     +c$lo12,c$ls3)
!        forall (i=i$l3:i$u3) c(i-c$lo3-l$b7+c$o) = c(i-c$lo3-l$b7+c$o) +
!    +c$f1(i-c$lo3-l$b+c$f1$o)
         c$lo3 = c$lo12
         do i$c12 = c$l3, c$u3, c$s3
            call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c12,i$l3,i$u3)
            do i = i$l3, i$u3
               c(i-c$lo3-l$b7+c$o) = c(i-c$lo3-l$b7+c$o) + c$f1(i-c$lo3-l$b
     ++c$f1$o)
            enddo
            c$lo3 = c$lo3 + c$ls3
         enddo
```

```
      call pghpf_comm_free(1,cp1)
      call pghpf_freen(1,c$f1$d(c$f1$d$o))
      call pgf90_deallocate(c$f1(c$f1$o),pghpf_0)
      call pghpf_comm_free(1,cp)
!     c(i-c$lo3-l$b7+c$o) = c(i-c$lo3-l$b7+c$o) + c$f1(i-c$lo3-l$b+
!    +c$f1$o)
!     endforall
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(28,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Seconda fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      continue
      continue
      cp2 = pghpf_comm_copy(c(c$o),a(a$o),c$d(c$d$o),a$d(a$d$o))
      xfer2 = pghpf_comm_start(cp2,c(c$o),c$d(c$d$o),a(a$o),a$d(a$d$o))
      call pghpf_comm_finish(xfer2)
      call pghpf_template(c$f3$dp,1,53248,c$d(c$d$o),0,1,100000)
      call pghpf_ptr_offset(c$f3$d$o,c$f3$dp,c$f3$d,25)
      call pghpf_instance(c$f3$dp,c$f3$d(c$f3$d$o),27,4,0,0,0)
      call pghpf_ptr_offset(c$f3$d$o,c$f3$dp,c$f3$d,25)
      call pghpf_allobnds(c$f3$d(c$f3$d$o),l$b1,u$b1)
      call pgf90_alloc(u$b1-l$b1+1,27,pghpf_0,c$f3$p,c$f3$o,c$f3)
      cp3 = pghpf_comm_copy(c$f3(c$f3$o),b(b$o),c$f3$d(c$f3$d$o),b$d(
     +b$d$o))
      xfer3 = pghpf_comm_start(cp3,c$f3(c$f3$o),c$f3$d(c$f3$d$o),b(b$o),
     +b$d(b$d$o))
      call pghpf_comm_finish(xfer3)
      call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l4,c$u4,c$s4,
     +c$lo13,c$ls4)
!     forall (ndi$vt=i$l4:i$u4:1) c(ndi$vt-c$lo4-l$b7+c$o) = c(ndi$vt-
!    +c$lo4-l$b7+c$o) + c$f3(ndi$vt-c$lo4-l$b1+c$f3$o)
      c$lo4 = c$lo13
      do i$c13 = c$l4, c$u4, c$s4
         call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c13,i$l4,i$u4)
         do ndi$vt = i$l4, i$u4
            c(ndi$vt-c$lo4-l$b7+c$o) = c(ndi$vt-c$lo4-l$b7+c$o) + c$f3(
     +ndi$vt-c$lo4-l$b1+c$f3$o)
         enddo
         c$lo4 = c$lo4 + c$ls4
      enddo
      call pghpf_comm_free(1,cp3)
      call pghpf_freen(1,c$f3$d(c$f3$d$o))
      call pgf90_deallocate(c$f3(c$f3$o),pghpf_0)
      call pghpf_comm_free(1,cp2)
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(37,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Terza fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      cp4 = pghpf_comm_copy(c(c$o),a(a$o),c$d(c$d$o),a$d(a$d$o))
      xfer4 = pghpf_comm_start(cp4,c(c$o),c$d(c$d$o),a(a$o),a$d(a$d$o))
      call pghpf_comm_finish(xfer4)
      call pghpf_template(c$f5$dp,1,53248,c$d(c$d$o),0,1,100000)
      call pghpf_ptr_offset(c$f5$d$o,c$f5$dp,c$f5$d,25)
      call pghpf_instance(c$f5$dp,c$f5$d(c$f5$d$o),27,4,0,0,0)
      call pghpf_ptr_offset(c$f5$d$o,c$f5$dp,c$f5$d,25)
      call pghpf_allobnds(c$f5$d(c$f5$d$o),l$b2,u$b2)
      call pgf90_alloc(u$b2-l$b2+1,27,pghpf_0,c$f5$p,c$f5$o,c$f5)
      cp5 = pghpf_comm_copy(c$f5(c$f5$o),b(b$o),c$f5$d(c$f5$d$o),b$d(
     +b$d$o))
      xfer5 = pghpf_comm_start(cp5,c$f5(c$f5$o),c$f5$d(c$f5$d$o),b(b$o),
```

```
     +b$d(b$d$o))
      call pghpf_comm_finish(xfer5)
      call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l5,c$u5,c$s5,
     +c$lo14,c$ls5)
!     forall (i$i=i$l5:i$u5:1) c(i$i-c$lo5-l$b7+c$o) = c(i$i-c$lo5-l$b7+
!    +c$o) + c$f5(i$i-c$lo5-l$b2+c$f5$o)
      c$lo5 = c$lo14
      do i$c14 = c$l5, c$u5, c$s5
         call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c14,i$l5,i$u5)
         do i$i = i$l5, i$u5
            c(i$i-c$lo5-l$b7+c$o) = c(i$i-c$lo5-l$b7+c$o) + c$f5(i$i-
     +c$lo5-l$b2+c$f5$o)
         enddo
         c$lo5 = c$lo5 + c$ls5
      enddo
      call pghpf_comm_free(1,cp5)
      call pghpf_freen(1,c$f5$d(c$f5$d$o))
      call pgf90_deallocate(c$f5(c$f5$o),pghpf_0)
      call pghpf_comm_free(1,cp4)
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(43,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Quarta fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      continue
!     forall (i=1:100000)
      cp6 = pghpf_comm_copy(c(c$o),b(b$o),c$d(c$d$o),b$d(b$d$o))
      xfer6 = pghpf_comm_start(cp6,c(c$o),c$d(c$d$o),b(b$o),b$d(b$d$o))
      call pghpf_comm_finish(xfer6)
      call pghpf_template(c$f7$dp,1,53248,c$d(c$d$o),0,1,100000)
      call pghpf_ptr_offset(c$f7$d$o,c$f7$dp,c$f7$d,25)
      call pghpf_instance(c$f7$dp,c$f7$d(c$f7$d$o),27,4,0,0,0)
      call pghpf_ptr_offset(c$f7$d$o,c$f7$dp,c$f7$d,25)
      call pghpf_allobnds(c$f7$d(c$f7$d$o),l$b3,u$b3)
      call pgf90_alloc(u$b3-l$b3+1,27,pghpf_0,c$f7$p,c$f7$o,c$f7)
      a$sp = pghpf_sect(a$d(a$d$o),100000,1,-1,1)
      call pghpf_ptr_offset(a$sp,a$sp,a$s,25)
      cp7 = pghpf_comm_copy(c$f7(c$f7$o),a(a$o),c$f7$d(c$f7$d$o),a$s(
     +a$sp))
      xfer7 = pghpf_comm_start(cp7,c$f7(c$f7$o),c$f7$d(c$f7$d$o),a(a$o),
     +a$s(a$sp))
      call pghpf_comm_finish(xfer7)
      call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l6,c$u6,c$s6,
     +c$lo15,c$ls6)
!     forall (i=i$l6:i$u6) c(i-c$lo6-l$b7+c$o) = c(i-c$lo6-l$b7+c$o) +
!    +c$f7(i-c$lo6-l$b3+c$f7$o)
      c$lo6 = c$lo15
      do i$c15 = c$l6, c$u6, c$s6
         call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c15,i$l6,i$u6)
         do i = i$l6, i$u6
            c(i-c$lo6-l$b7+c$o) = c(i-c$lo6-l$b7+c$o) + c$f7(i-c$lo6-
     +l$b3+c$f7$o)
         enddo
         c$lo6 = c$lo6 + c$ls6
      enddo
      call pghpf_comm_free(1,cp7)
      call pghpf_free(a$s(a$sp))
      call pghpf_freen(1,c$f7$d(c$f7$d$o))
      call pgf90_deallocate(c$f7(c$f7$o),pghpf_0)
      call pghpf_comm_free(1,cp6)
!     c(i-c$lo6-l$b7+c$o) = c(i-c$lo6-l$b7+c$o) + c$f7(i-c$lo6-l$b3+
!    +c$f7$o)
!     endforall
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(51,'prova.hpf')
```

```
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Quinta fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      continue
!     forall (i=1:100000)
      a$sp1 = pghpf_sect(a$d(a$d$o),100000,1,-1,1)
      call pghpf_ptr_offset(a$sp1,a$sp1,a$s1,25)
      cp8 = pghpf_comm_copy(d(d$o),a(a$o),d$d(d$d$o),a$s1(a$sp1))
      xfer8 = pghpf_comm_start(cp8,d(d$o),d$d(d$d$o),a(a$o),a$s1(a$sp1))
      call pghpf_comm_finish(xfer8)
      call pghpf_cyclic_loop(d$d(d$d$o),1,1,100000,1,c$l7,c$u7,c$s7,
     +c$lo16,c$ls7)
      call pghpf_comm_free(1,cp8)
      call pghpf_free(a$s1(a$sp1))
!     d(i-c$lo7-l$b8+d$o) = d(i-c$lo7-l$b8+d$o)
!     endforall
      continue
!     forall (i=1:100000)
      cp9 = pghpf_comm_copy(c(c$o),b(b$o),c$d(c$d$o),b$d(b$d$o))
      xfer9 = pghpf_comm_start(cp9,c(c$o),c$d(c$d$o),b(b$o),b$d(b$d$o))
      call pghpf_comm_finish(xfer9)
      call pghpf_template(c$f9$dp,1,53248,c$d(c$d$o),0,1,100000)
      call pghpf_ptr_offset(c$f9$d$o,c$f9$dp,c$f9$d,25)
      call pghpf_instance(c$f9$dp,c$f9$d(c$f9$d$o),27,4,0,0,0)
      call pghpf_ptr_offset(c$f9$d$o,c$f9$dp,c$f9$d,25)
      call pghpf_allobnds(c$f9$d(c$f9$d$o),l$b4,u$b4)
      call pgf90_alloc(u$b4-l$b4+1,27,pghpf_0,c$f9$p,c$f9$o,c$f9)
      cp10 = pghpf_comm_copy(c$f9(c$f9$o),d(d$o),c$f9$d(c$f9$d$o),d$d(
     +d$d$o))
      xfer10 = pghpf_comm_start(cp10,c$f9(c$f9$o),c$f9$d(c$f9$d$o),d(d$o
     +),d$d(d$d$o))
      call pghpf_comm_finish(xfer10)
      call pghpf_cyclic_loop(c$d(c$d$o),1,1,100000,1,c$l8,c$u8,c$s8,
     +c$lo17,c$ls8)
!     forall (i=i$l8:i$u8) c(i-c$lo8-l$b7+c$o) = c(i-c$lo8-l$b7+c$o) +
!    +c$f9(i-c$lo8-l$b4+c$f9$o)
      c$lo8 = c$lo17
      do i$c17 = c$l8, c$u8, c$s8
         call pghpf_block_loop(c$d(c$d$o),1,1,100000,1,i$c17,i$l8,i$u8)
         do i = i$l8, i$u8
            c(i-c$lo8-l$b7+c$o) = c(i-c$lo8-l$b7+c$o) + c$f9(i-c$lo8-
     +l$b4+c$f9$o)
         enddo
         c$lo8 = c$lo8 + c$ls8
      enddo
      call pghpf_comm_free(1,cp10)
      call pghpf_freen(1,c$f9$d(c$f9$d$o))
      call pgf90_deallocate(c$f9(c$f9$o),pghpf_0)
      call pghpf_comm_free(1,cp9)
!     c(i-c$lo8-l$b7+c$o) = c(i-c$lo8-l$b7+c$o) + c$f9(i-c$lo8-l$b4+
!    +c$f9$o)
!     endforall
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(62,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Sesta fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      call pghpf_redistribute(a$dp,1,33792,0,-1,1,100000)
      call pghpf_ptr_offset(a$d$o,a$dp,a$d,25)
      call pghpf_redistribute(b$dp,1,33792,0,-1,1,100000)
      call pghpf_ptr_offset(b$d$o,b$dp,b$d,25)
      call pghpf_redistribute(c$dp,1,33792,0,-1,1,100000)
      call pghpf_ptr_offset(c$d$o,c$dp,c$d,25)
```

```
      call pghpf_redistribute(d$dp,1,33792,0,-1,1,100000)
      call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(68,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Settima fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
      call pghpf_sysclk(tp,tr,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(tp,tr)
      call pghpf_redistribute(d$dp,1,33792,0,0,1,100000)
      call pghpf_ptr_offset(d$d$o,d$dp,d$d,25)
      call pghpf_sysclk(ta,pghpf_0,pghpf_0,pghpf_1,pghpf_1,pghpf_1)
!     call system_clock(ta)
      call pgf90io_src_info(74,'prova.hpf')
      z__io = pgf90io_ldw_init(6,0,0,0)
      z__io = pgf90io_ldw(14,1,0,'Ottava fase: ')
      z__io = pgf90io_ldw(25,1,0,ta - tp)
      z__io = pgf90io_ldw_end()
99999 continue
      call pghpf_freen(5,sp$d(sp$dp),a$d(a$d$o),b$d(b$d$o),c$d(c$d$o),
     +d$d(d$d$o))
      call pgf90_deallocate(d(d$o),pghpf_0)
      call pgf90_deallocate(c(c$o),pghpf_0)
      call pgf90_deallocate(b(b$o),pghpf_0)
      call pgf90_deallocate(a(a$o),pghpf_0)
      end
```

Bibliografia

[Bra96a] Thomas Brandes. *ADAPTOR: Distributed Array Library (DALIB) Ver. 4.0*, Aprile 1996.

[Bra96b] Thomas Brandes. *ADAPTOR: Installation Guide Ver. 4.0*, Aprile 1996.

[Bra96c] Thomas Brandes. *ADAPTOR: Programmer's Guide Ver. 4.0*, Aprile 1996.

[Bra96d] Thomas Brandes. *ADAPTOR: Users guide Ver. 4.0*, Aprile 1996.

[CCL95] B. Cantalupo, P. Criscione, and D. Laforenza. Introduzione al sistema ibm sp2 e al suo ambiente di programmazione parallela. Technical report, CNUCE, Rapporto Interno C95-38, 5 Dicembre 1995.

[Cor96] Digital Equipment Corporation. *Digital High Performance Fortran 90. HPF and PSE Manual*, 1996.

[DMS96] Jack Dangarra, Philip Mucci, and Eric Strohmaier. Parkbench 2.0: Release notes and run rules. Technical report, 13 Maggio 1996.

[For94] High Performance Fortran Forum. High Performance Fortran language specification ver. 1.1. Technical report, Rice University, Nov. 1994.

[Groa] The Portland Group. pghpf *: Profiler User's Guide.*

[Grob] The Portland Group. pghpf *: Reference Manual.*

[Groc] The Portland Group. pghpf *: User's Guide.*

[HB94] Roger Hockney and Michael Berry. Public international benchmarks for parallel computers. Technical report, PARKBENCH Committee, 7 febbraio 1994.

[HRV95] T. Haupt, S.G. Reddy, and G. Vengurlekar. Low level HPF compiler benchmark suite. Technical report, Northeast Parallel Architectures Center at Syracuse University, Agosto 1995.

[IBMa] IBM. *IBM AIX Parallel Environment: Programming Primer*, release 2.1 edition.

[IBMb] IBM. *LoadLeveler: User's Guide.*

[IBM96] IBM. *XL High Performance Fortran for AIX. Language Reference*, version 1 release 1 edition, 1996.

[SA96] Maria Chiara Sechi and Michele Aiello. *Estensioni al paradigma data parallel HPF per la ridistribuzione in ambienti di calcolo eterogenei.* PhD thesis, Università degli Studi di Pisa, 1996.

[SAH+] Craig B. Stunkel, Mark Atkins, Peter H. Hochschild, Richard A. Swetz, Dennis G. Shea, Carl A. Bender, Douglas J. Josef, Robert F. Stucke, Philip R. Varker, Bülent Abali, Don G. Grice, Ben J. Nathanson, and Michael Tsao. The SP2 Communication Subsystem. 1994.

[Sura] Meiko World Incorporated Computing Surface. *CS-2 Hardware Overview.*

[Surb] Meiko World Incorporated Computing Surface. *Documentation Guide.*

[Surc] Meiko World Incorporated Computing Surface. *Getting Started – User's Guide.*

[Surd] Meiko World Incorporated Computing Surface. *TotalView Supplement for Meiko CS-2 Users.*

[Sure] Meiko World Incorporated Computing Surface. *Vector Processing Element Overview.*