ハイパフォーマンス
コンピューティング 79－8
（1999．12．3）

異機種並列分散システムにおけるHPFプログラムの実行と評価

荒木拓也† 松浦健一郎† 村井均†
末広謙二† 小西弘一† 妹尾義樹†

近年，数値計算アプリケーションは大規模化，複雑化の一途をたどっている．このようなプログラムを効率的に実行するためには，ベクトル並列計算機やスカラ並列計算機のような異なった性質を持つ計算機をネットワークで結合し，それらを一つの計算機システムとして扱う，異機種並列分散システムが有望であると考えられている．しかし，既存の技術で異機種並列分散システムを利用しようとした場合，システムに存在するさまざまな「シーム」により，プログラムが非常に複雑になるという問題点があった．

本研究では，ベクトル並列計算機SX-4とスカラ並列計算機Cenju-4を接続した異機種並列分散システムを構成し，それらをシームレスに結合する異機種MPIライブラリを構築した．さらに，この異機種MPIを利用する形でHPFコンパイラを構築し，HPFプログラムから異機種並列分散システムをシームレスに利用可能にした．

このシステム上でHPFプログラムの実行，評価を行ない，HPFプログラムが異機種並列分散システムの性能を十分に引き出していることを示した．

Execution and evaluation of HPF programs on a heterogeneous parallel and distributed system

Takuya Araki,† Ken-ichiro Matsuura,† Hitoshi Murai,†
Kenji Suehiro,† Koichi Konishi† and Yoshiki Seo †

In recent years, numerical applications are getting larger and more complex. In order to execute these applications efficiently, heterogeneous parallel and distributed systems which connect various kinds of computers like scalar-parallel computers and vector-parallel computers are considered promising. However, utilizing such systems with existing technology is difficult because various "seams" within a system make the programming quite complex.

In this study, we implemented a heterogeneous MPI library which seamlessly handles communication in a heterogeneous parallel and distributed system that is composed of a vector-parallel computer SX-4 and a scalar-parallel computer Cenju-4. In addition, we implemented an HPF compiler on top of that, which makes it possible to use the heterogeneous system seamlessly through the HPF programming.

We executed and evaluated HPF programs on the system and showed that these HPF programs fully extracts the ability of the heterogeneous parallel and distributed system.

1. はじめに

コンピュータシステムの急速な性能向上により，航空・宇宙・原子力・地球科学・分子科学・気象など，さまざまな分野で数値シミュレーションが利用されるようになった．

こういった数値シミュレーションは，これまで単一の物理現象を対象にした比較的単純なものが中心であった．しかし，今後は複数の物理現象が複雑に絡み合った大規模なシミュレーションが行なわれるようになり，演算性能に対する要求や多種多様な計算特性に対する要求が一段と高まると考えられる．

我々はこのような要求に応えるため，ベクトル並列計算機とスカラ並列計算機，あるいは共有メモリ並列計算機と分散メモリ並列計算機のような，異なる特徴を持つ計算機をネットワークで結合した異機種並列分散システムの研究を行なっている．このようなシステムでは，

- 計算資源を結合することでそれらを有効に利用できる
- それぞれのシステムが得意とする計算を実行できる

という特徴がある．後者の特徴により，例えば複雑な数値シミュレーションプログラムのうち，スカラ並列計算機に向いた計算はスカラ並列計算機で，ベクトル並列計算機に向いた計算はベクトル並列計算機で実行すること

† 新情報処理開発機構 並列分散システムNEC研究室
Parallel and Distributed Systems NEC Laboratory, RWCP

で, 両者の特徴を生かした高速な計算が実現できると期待される.

しかし, 従来技術では異機種並列分散システム向けのプログラムを開発することは容易ではない. これは, 各システムの性能差や各システムに渡る階層的な並列性を考慮に入れた上で, それぞれのシステムで別々のプログラムを作成しなければならないためである.

我々はこの問題を解決するため, 異機種並列分散システム向けのシームレスなプログラミング環境の構築に取り組んでいる[8),9)]. このシステムは, 異機種並列分散システム向けの異機種 MPI ライブラリやコンパイラの他, 並列分散化を支援するツールや実行時モニタリングシステムから構成される.

本研究ではこのプログラミング環境の一部である, 異機種 MPI ライブラリ, HPF コンパイラを用いた.

異機種 MPI ライブラリはベクトル並列計算機 SX-4 とスカラ並列計算機 Cenju-4 を結合した異機種並列分散システム上で動作し, 結合されたシステムを一つの並列計算機として利用することができる.

また, HPF コンパイラはこの異機種 MPI ライブラリを利用する形で構築している. このコンパイラにより, HPF プログラムから異機種並列分散システムをシームレスに利用可能になる.

本研究ではこれらのライブラリ, コンパイラを用いて, 異機種並列分散システムにおいて HPF プログラムの実行, 評価を行なった.

本稿の構成は以下の通りである. まず第 2節で, 対象とする異機種並列分散システムとして, システムのハードウェア, 異機種 MPI ライブラリ, HPF コンパイラについて述べる. 第 3節で HPF プログラムによる本システムの評価について述べる. 第 4節で今後の課題について述べ, 第 5節で関連研究について述べた後, 第 6節でまとめを行う.

2. 異機種並列分散システム

2.1 ハードウェア

本研究で扱う異機種並列分散システムは, 共有メモリ型ベクトルスーパーコンピュータ SX-4 と分散メモリ型スカラ並列計算機 Cenju-4 をネットワーク (現在は 10Base-T の LAN) で結合したものである (図 1). ネットワークは, 将来的にはより高速なものに置き換える予定である. 本研究で用いたモデルの諸元については, 表 1 にまとめる.

2.2 異機種 MPI ライブラリ

本研究で用いた MPI ライブラリはベクトル並列計算機 SX-4 とスカラ並列計算機 Cenju-4 を結合した異機種並列分散システム上で動作し, 結合されたシステムを一つの並列計算機として MPI ライブラリから利用することができる.

すなわち, 例えば SX-4 のプロセッサをプロセッサ番号 1 番, 2 番, Cenju-4 のプロセッサをプロセッサ番号 3 番, 4 番... などとして利用することができる*. これにより, ネットワークで結合された複数の並列計算機を一つの並列計算機としてシームレスに利用することが可能である.

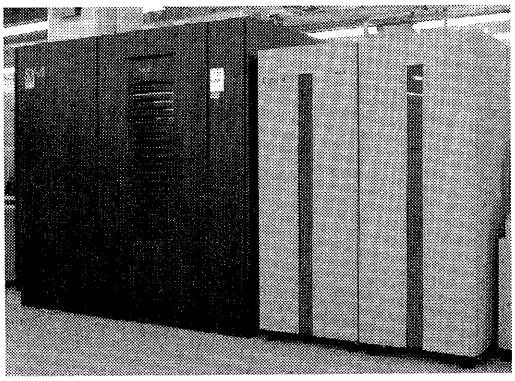

図 1 異機種並列分散システム

表 1 ハードウェア諸元

	SX-4	Cenju-4
プロセッサ台数	2	32
プロセッサ	—	VR10000 (200MHz)
ピーク性能 (1CPU)	2G FLOPS	400M FLOPS
プロセッサ間通信速度	(共有メモリ)	200MByte/s ×双方向

表 2 に, 本 MPI ライブラリの性能評価を示す. これは, SX-4 のプロセッサと Cenju-4 のプロセッサ間の通信のスループットとレイテンシを示すものである. ここで, スループットは転送サイズを 1KByte から 10MByte までを変えて測定した. また, レイテンシは 4 バイトのデータを往復するのにかかった時間を 2 で割ったものである.

現在は SX-4, Cenju-4 間は 10Base-T の LAN で結合されているため, 計算速度に比較するとネットワークの性能は低い.

表 2 SX-4, Cenju-4 間のネットワーク性能

スループット (KByte/sec)					レイテンシ
1KB	10KB	100KB	1MB	10MB	
519	787	786	788	796	1.74 msec

2.3 HPF コンパイラ

HPF コンパイラは, 第 1節で述べた通り, 異機種並列分散システム向けのシームレスなプログラミング環境の一部として開発中のものである.

* 厳密には MPI におけるプロセッサ番号 (ランク) は 0 から始まるが, 説明を簡単にするため, 本稿ではプロセッサ番号は 1 番から用いる.

本コンパイラは, 前節で述べた異機種 MPI ライブラリを利用することで, 異機種並列分散環境を HPF プログラムからシームレスに利用することが可能である.

また, 本コンパイラは HPF1.1 の規格にある HPF 指示行を受け付ける他, ループの各繰り返しを明示的にプロセッサに割り付ける, イタレーション・マッピングと呼ばれる拡張機能を持つ[7]. この機能により, 例えば, 繰り返しの1番目はプロセッサ1番で実行し, 2番目はプロセッサ2番で実行する, というような指示が可能になる.

さらに, 異機種並列分散環境でのプログラミングを容易に行なうための拡張機能を実装中である. ただし, こちらの機能は現在実装中であるため, 評価には用いていない. この機能については第4.1で詳しく述べる.

3. 評　　価

以上の環境で HPF プログラムを実行, 評価した. 以下にプログラムの説明と並列化手法, 評価結果を述べる.

3.1 EP

3.1.1 プログラム

EP は NAS Parallel Benchmarks に収められたベンチマークプログラムの一つである. 今回は NAS Parallel Benchmarks 2.3 版に収められた逐次版 EP を元にした☆.

プログラムは以下のような形で構成されている:

```
      do 150 k = 1, np
         ....
         call vranlc(2 * nk, t1, a, x)
         do 140 i = 1, nk
            ....
               q(l) = q(l) + 1.d0
140      continue
150   continue
```

この "do 150" のループがほとんどの実行時間を占める. このループは各繰り返し間に依存関係がなく, 完全に独立に計算できる.

このループの中では "`vranlc(2 * nk, t1, a, x)`" によって乱数の生成がおこなわれ, その値を用いて "do 140" のループで計算が行なわれる. 計算結果は q に保存される.

乱数生成部は NAS Parallel Benchmarks の 2.3 版に付属のベクトル化可能な版を用いた.

また, "do 140" のループは, 配列 q の添字 l の値が計算によって決定するため, そのままではベクトル化できなかった. このため, "do 140" のループでは l の値を配列に保存するように変更することでベクトル化を可能にし, q への反映は別のループで行なうようにした.

3.1.2 並列化手法

まず, "do 150" のループを並列化するため, このループの外側にプロセッサ台数分だけ繰り返す新たなループを付け加える. このループに対して `!HPF$ independent` を指定し, かつ第2.3節で述べたイタレーション・マッピング機能を用いて, 各繰り返しが各々のプロセッサで実行されるように指定した. ここで, "do 150" の繰り返しをプロセッサごとに分割して実行する必要があるため, (ループカウンタで表された) プロセッサ番号によって, k の開始値, 終了値を設定する.

ここで, SX-4 と Cenju-4 は単体 CPU の速度が異なるため, 適切な負荷分散を行なわなければ性能が低下する. したがって, この "do 150" の繰り返しを, CPU1 台の速度が速い SX-4 では多く, 遅い Cenju-4 では少なく実行するように k の開始値, 終了値を設定した. 割り当ての比率は単体での実行時間に基づいて, SX-4 : Cenju-4 が 9 : 1 になるようにした.

また, 計算結果を各プロセッサでローカルに保存するため, 配列 q の次元を拡張した2次元配列 qg を新たに宣言した. プロセッサ1番は `qg(:,1)` を, プロセッサ2番は `qg(:,2)` を使うなどとして, 繰り返しの中では qg にたいして結果を保存する. 計算結果は終了後に q に集計するようにした.

3.1.3 評 価 結 果

図 2 に相対速度のグラフを, **表 3** に実行時間の表を示す. データサイズは "Class A" を用いた.

グラフの Z 軸は (並列化したプログラムを)Cenju-4 の 1CPU で実行した場合の速度を 1 とした相対速度を示す. X 軸, Y 軸は Cenju-4 の CPU 台数, SX-4 の CPU 台数である (Cenju-4 の台数は異機種 MPI の現在の実装上の制限により 8 台までである). 各軸で CPU 台数が 0 の部分は, どちらかの計算機単体での実行結果である. この場合, SX-4, Cenju-4 間の負荷分散をしないで良いため, 負荷を均等に分割する別プログラムを用意し, オーバヘッドを減らして評価した.

また, Serial の部分には, 並列化せずに Fortran コンパイラでコンパイル, 実行した場合の速度を示した.

HPF コンパイラは, 第2.3節で述べたものを用いた. HPF コンパイラは MPI を用いて並列化した Fortran プログラムを出力する. この Fortran プログラムは SX-4, Cenju-4 共にシステムに標準の f90 コンパイラを用いてコンパイルした. コンパイラのオプションは, HPF コンパイラは, SX-4, Cenju-4 共 `-O3 -R5`, Fortran コンパイラは, SX-4 は `-C hopt`, Cenju-4 は `-KGnum=0 -O -Zstatic -Kmips2` である.

実行時間は3回計測し, 大きなばらつきが無いことを確認して平均を取った.

グラフを見ると, SX-4 単体, Cenju-4 単体での実行速度は, 台数が増えると共にほぼ理想的に向上している. また, SX-4, Cenju-4 の両者を用いた場合も, ほぼそれぞれの実行速度の和になっており, 理想に近い速度向上を達成している.

また, 並列化を行なわない場合と並列化を行なって

☆ プログラムが異なるため, SX-4 単体のベンチマーク結果として公開されている値とは結果が異なる.

表3 EP の実行時間 (秒)

Cenju-4		0	1	2	3	4	5	6	7	8
SX-4	0	–	500.9	249.4	167.4	125.3	102.1	83.8	71.9	62.2
	1	53.8	49.3	45.3	42.1	38.7	35.7	33.4	31.7	29.4
	2	27.1	25.9	25.0	24.4	23.5	22.5	21.9	20.8	20.2

	逐次実行
Cenju-4	496.6
SX-4	52.9

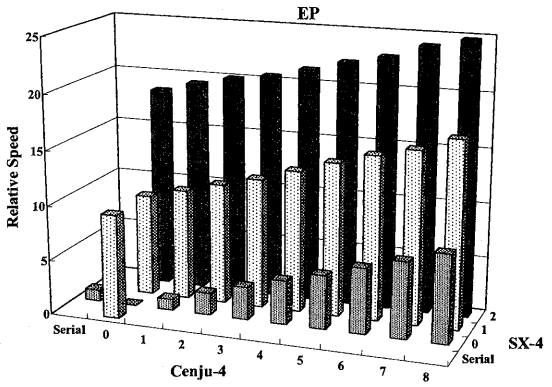

図2 EP の相対速度

1台で実行した場合の実行時間を比較すると, SX-4, Cenju-4 共にわずかな差であり, 並列化に伴うオーバヘッドは小さいことがわかる.

SX-4 の CPU1 台で並列化したプログラムを実行した時間が 53.8 秒であり, Cenju-4 の CPU1 台で並列化したプログラムを実行した時間は 500.9 秒であるから, 両者の速度比は約 9:1 である. したがって, Cenju-4 の CPU を 8 台, SX-4 の CPU を 2 台で実行した場合の理想的な実行時間は, $500.9 \times \frac{1}{9 \times 2 + 8}$ で 19.3 秒である. 実際の実行時間 20.2 秒は, 若干の差はあるものの, これに近い理想的なものになっている.

3.2 grid

3.2.1 プログラム

grid は APR Benchmark Suite に収められたベンチマークプログラムの一つである. プログラムは 2 次元配列上で緩和計算を行なうものであり, 近傍のデータを参照する必要がある. 計算の中心部分は以下のようなコードになっている:

```
do it=1,ncycles
  do j=2,n+1
    do i=2,n+1
      grid(i,j,inew)=
        exp((alog(grid(i-1,j-1,iold))+
     .    alog(grid(i-1,j,iold))+
          alog(grid(i-1,j+1,iold))+
          alog(grid(i,j-1,iold))+
          alog(grid(i,j,iold))+
          alog(grid(i,j+1,iold))+
          alog(grid(i+1,j-1,iold))+
          alog(grid(i+1,j,iold))+
          alog(grid(i+1,j+1,iold)))/9.0)
    end do
  end do
  ....
end do
```

計算は配列 grid 上で行なわれる. grid は 3 次元配列であるが, 3 次元目の大きさは 2 であり, 2 次元配列を 2 つ持つ形になっている. この計算は ncycles 回繰り返され, 1 回毎に 3 次元目の添字 iold と inew の値が入れ換えられる.

並列化のため, grid を 2 次元目で block 分割したとすると, 計算前に隣のプロセッサが持っているデータを通信により取得する必要がある. また, このプログラムでは 2 次元配列をトーラスとして扱っており, この計算後に明示的に配列の上下, 左右をお互いにコピーする. このため, プログラム全体の通信は, 一回の繰り返しで, 4 回 2 次元配列の 1 列分を通信することになる.

3.2.2 並列化手法

SX-4, あるいは Cenju-4 単体で並列実行する場合は, grid を 2 次元目で block 分割し, "do j=2,n+2" のループに "!HPF$ independent" を挿入することで, で並列化できる. しかし, そのままで SX-4 と Cenju-4 の異機種並列環境で実行すると, 単体 CPU の性能の違いから負荷分散が悪くなり性能が悪化する.

したがって, grid の 2 次元目を以下のように次元拡張することで, 負荷分散を行なった.

簡単のため, 1 次元で説明する. 1 次元配列の長さを L とし, SX-4 の単体 CPU の速度が Cenju-4 の単体 CPU の速度の N 倍速いとする. また, SX-4 の CPU 数を NP_{SX}, Cenju-4 の CPU 数を NP_{Cenju} とする. ここで, 仮想的な CPU の数を $PNP = NP_{SX} \times N + NP_{Cenju}$ とすると, $\frac{L}{PNP}$ が Cenju-4 に割り当てるべき要素数で, $\frac{L}{PNP} \times N$ が SX-4 に割り当てるべき要素数となる.

そこで, $\frac{L}{PNP} \times N$ と $NP = NP_{SX} + NP_{Cenju}$ を各次元の長さとする 2 次元配列を確保し, NP の次元で block 分割する (図 3).

ここで, SX-4 では配列の長さを $\frac{L}{PNP} \times N$, Cenju-4 では $\frac{L}{PNP}$ であるとして扱うことで, 負荷分散を行なうことができる.

ただし, 本プログラムの場合, 分割次元で隣の要素の値を計算に必要とする. したがって, "do j=2,n+1" のループに入る前に, 明示的に要素のコピーを行なうコードを挿入した.

また, 今回の測定では, 単体での実行時間に基づいて,

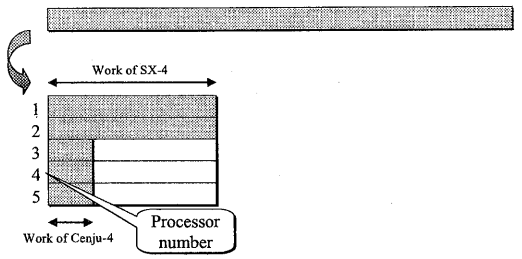

図3 負荷分散手法

SX-4 の単体 CPU の速度が Cenju-4 の単体 CPU の速度の 15 倍であるとして, 配列の分割を行なった.

3.2.3 評価結果

図4 に相対速度のグラフを, 表4 に実行時間の表を示す.

配列のサイズは 2000 × 2000 とし, 繰り返しの回数は 20 とした. 後で述べる解析を簡単にするため, 計時は "`do it=1,ncycles`" のループの終りまでとし, 結果集計部分は含めなかった.

グラフは, EP の場合と同様, Z 軸は (並列化したプログラムを)Cenju-4 の 1CPU で実行した場合の速度を 1 とした相対速度を示す. X 軸, Y 軸は Cenju-4 の CPU 台数, SX-4 の CPU 台数である. ただし, SX-4 を 2CPU で実行した場合は, 実行時間のばらつきが大きかったため, 示していない.

その他の測定条件は EP の場合と同様である.

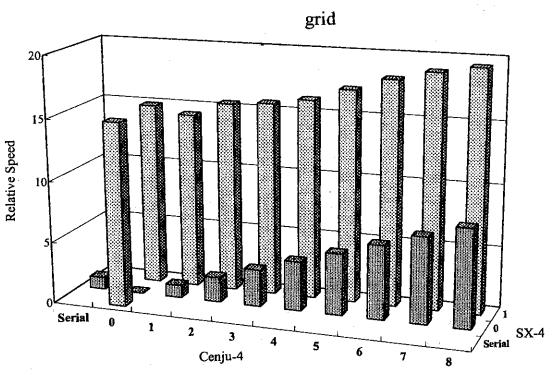

図4 grid の相対速度

まず, 並列化を行なわない場合と並列化を行なって 1 台で実行した場合の実行時間を比較すると, EP の場合と同様, SX-4, Cenju-4 共にわずかな差であり, 並列化に伴うオーバヘッドは小さいことがわかる.

また, グラフを見ると, SX-4 1CPU に対して Cenju-4 1CPU を付加した場合, SX-4 1CPU の場合に比べて速度が低下している. その後, Cenju-4 の CPU 数を増やしていく毎に性能が向上している.

これを実行時間から解析すると以下のようになる. プログラムの実行中に行なわれる通信の量は, 2000 要素 × 4Byte × 4 回×繰り返し 20 回で, 640000Byte である. 一回の通信は 2000 要素 × 4Byte であるから, 約 8KByte である.

Cenju-4 内, SX-4 内の通信速度は Cenju-4, SX-4 間の通信速度に比べると非常に速いので, その通信時間は無視し, Cenju-4, SX-4 間の通信時間について考える. 表2により, そのスループットを 780KByte/sec とすると, プログラムの 1 回の実行にかかる通信時間は約 $\frac{640000}{780*1024}$ = 0.8 秒である. 通信のレイテンシは 1.74 msec で, 通信回数は 4 × 20 回であるから, 通信起動にかかる時間は 0.14 秒程度である. したがって, SX-4, Cenju-4 間の通信で, 最低 0.94 秒程度の時間がかかることになる.

SX-4 1CPU の実行時間が 15.4 秒で, SX-4 1CPU の速度が Cenju-4 1CPU の速度の 15 倍であるとすると, SX-4 1CPU, Cenju-4 1CPU での, 通信時間を除いた理想的な実行時間は, $15.4 \times \frac{15}{15+1} = 14.4$ 秒である. これに 0.94 を加えると約 15.3 秒となる. 実際の実行時間 16.0 秒はそれよりもやや長めにかかっているが, 負荷分散のためのオーバヘッドや, 速度比が正確に 15 : 1 ではないことに起因すると考えられる.

SX-4 1CPU, Cenju-4 8CPU の場合は, 理想的な実行時間は $15.4 \times \frac{15}{15+8} + 0.94 = 11.0$ 秒となり, 実際の実行時間 11.8 秒と比べると, 先ほどと同様の若干の差を除けば, ほぼ計算通りの実行時間となっている.

4. 今後の課題

4.1 異機種並列分散実行用拡張機能の実装と評価

今回の評価では, HPF1.1 に若干拡張された機能のみを用いて並列化を行なったため, 負荷分散を行なうために, 直截的ではないプログラムの変更が必要になった. また, プログラムが複雑になると, 今回評価に使った方法での並列化には限界がある.

今回用いたような簡単なプログラムの場合は, HPF2.0 の公認拡張にある `GEN_BLOCK` を利用することで, より簡単に負荷分散を指定できる. しかし, `GEN_BLOCK` を用いても, SX-4 と Cenju-4 にどれだけの負荷を割り当てるかは, 実際に速度を測定し, 配列値の形でプログラム中に埋め込む必要がある.

このため, 我々は第 1節で述べたようなプログラミング環境として, プログラムの性能予測や負荷の分割などを GUI を用いて半自動的に行なう環境を構築している. この環境を実装し, プログラミング環境を含めた評価を, Cenju-4, SX-4 間のネットワークを高速なものに置き換えた上で行なうことが, 今後の課題にあげられる.

また, ベクトル並列計算機, スカラ並列計算機の特徴を利用した並列化を行なう場合, これだけでは不十分である. そこで, 我々は現在異機種並列分散実行用の拡張機能の実装を進めている. これは HPF2.0 の公認拡張にある, `ON`, `TASK_REGION` を用いるタスク並列の枠組を拡

表 4 grid の実行時間 (秒)

Cenju-4		0	1	2	3	4	5	6	7	8
SX-4	0	–	228.6	114.4	76.4	57.3	45.9	38.2	32.8	28.7
	1	15.4	16.0	14.8	14.6	14.2	13.3	12.6	12.1	11.8

	逐次実行
Cenju-4	235
SX-4	15.3

張する. この拡張は SUBSYSTEM 文, SUBDIVIDE 文から構成され, 以下のように記述する:

```
PROCESSORS :: PE(8)
SUBSYSTEM (SCALAR) :: CENJU
SUBSYSTEM (VECTOR) :: SX
SUBDIVIDE (/5, 3/) WITH (CENJU, SX) :: PE
```

この場合, SUBSYSTEM 文に SCALAR を指定することでスカラプロセッサとして CENJU を定義し, 同様に VECTOR を指定することでベクタプロセッサとして SX を定義している. さらに, SUBDIVIDE 文により, PROCESSORS 文で宣言された 8 つのプロセッサを CENJU と SX に分割している.

そして, ON 部に CENJU あるいは SX を指定することで, それぞれのシステムでの実行を指定する.

この枠組を使うことで, より明確に異機種システムの特徴を生かした並列実行が可能になる. 今後の課題として, この拡張機能を実装し, 下に述べる実アプリケーションでの評価を行なうことがあげられる.

4.2 実アプリケーションでの評価

今回評価に用いたプログラムは, 比較的単純なベンチマークプログラムである. しかし, 異機種並列分散システムは, より複雑な実アプリケーションの実行において, より効果を発揮すると考えられる.

このようなアプリケーションの例として, 粒子シミュレーションがあげられる. 粒子シミュレーションは, 通常各粒子の振舞いの計算と, 場の計算から構成される. 粒子の振舞いの計算は通信が不要なため, スカラ並列計算機 (あるいは, スカラ並列計算機 + ベクトル並列計算機) が適している. 一方場の計算は多くの通信を要するため, ベクトル並列計算機単体で実行した方が高速であると考えられる.

今後の課題として, このような実アプリケーションで異機種並列分散システムの評価を行なうことがあげられる.

5. 関連研究

分散環境で並列計算を行なうことを目的とした研究は数多くなされている. 我々は, 性質の異なった計算機の効率的な利用を, 言語レベルからシームレスに行なうことを目的としている.

これに対し, Ninf[4], NetSolve[3] はライブラリ層で分散計算の制御を行なうことを目的としている. また, Globus プロジェクト[1] は広域計算において通信, セキュリティ管理, 資源管理その他を行なうツールキットを提供している.

本研究で用いたような異機種間を接続する MPI ライブラリには, Globus ツールキットを用いる Grid-Enabled MPI[6] や, 複数の MPI 実装の接続を可能にする規格である IMPI[2] などがある.

データ並列言語にタスク並列性を導入する試みには, Fx, Opus などがあるが[5], いずれも異機種並列分散環境での実行を目標にしている我々の試みとは異なる.

6. おわりに

本研究では, ベクトル並列計算機 SX-4 とスカラ並列計算機 Cenju-4 を接続した異機種並列分散システムを構成し, その上で HPF プログラムの実行, 評価を行なった. この評価により, HPF プログラムが異機種並列分散システムの性能を十分に引き出していることを示した. また, これにより, 異機種並列分散システムの有効性を確認した.

参考文献

1) The Globus project. http://www-fp.globus.org/.
2) Interoperable MPI. http://impi.nist.gov/IMPI/.
3) NetSolve Homepage. http://www.cs.utk.edu/netsolve/.
4) Ninf: A Global Computing Infrastructure. http://ninf.etl.go.jp/.
5) HenriE. Bal and Matthew Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, Vol. 6, No. 3, pp. 74–84, 1998.
6) Ian Foster and Nicholas T. Karonis. A Grid-Enabled MPI: Message passing in heterogeneous distributed computing systems. In *SuperComputing 98*, 1998.
7) 蒲池恒彦, 草野和寛, 末廣謙二, 妹尾義樹, 田村正典, 左近彰一. HPF 処理系の実現と評価. 情報処理学会論文誌, Vol. 37, No. 7, pp. 1255–1264, July 1996.
8) 草野和寛, 妹尾義樹, 左近彰一, 片山博, 小藤雅俊, 中田登志之. 異機種並列分散処理による高性能計算機システムの構想. 電子情報通信学会技術研究報告, Vol. 97, No. 226, pp. 91–96, 1997. CPSY97-63.
9) 片山博, 中田登志之, 妹尾義樹, 左近彰一, 西川岳, 末広謙二. 高性能並列分散システムの設計. Technical Report TR-98001, RWC, 1998.