# 超並列計算機CP-PACSにおけるNPB Kernel CGの評価

板倉憲一[†] 松原正純[†] 朴泰祐[†]
中村宏[††] 中澤喜三郎[†††]

本論文ではNAS並列ベンチマークのkernel CGを用いて，超並列計算機CP-PACSの性能評価および解析を行い，CP-PACSにおける並列プログラムの最適化について述べる．Kernel CGでは，広範囲で複雑なデータ転送や短ベクトル処理が要求されるために並列化効率を上げることが難しいが，CP-PACSでは柔軟なハイパクロスバ網と擬似ベクトル処理機構によって最適化を行うことが可能である．評価に用いた並列プログラムは基本的な通信性能とCPU性能に基づいてFORTRANおよびアセンブラ・ソースでのチューニングを行い，CPU clock counterによる精密な実行時間測定を基に性能解析を行った．解析結果から通信時間とCPU処理時間の最適化にはトレードオフが生じるために，そのチューニングが必要であることが分かった．また，このような最適化を行うことで，他の並列計算機に対してCP-PACSが高い台数効果と絶対性能を持つことを示すことができた．

# Performance Evaluation of NPB Kernel CG on CP-PACS

Ken'ichi Itakura,[†] Masazumi Matsubara,[†] Taisuke Boku,[†]
Hiroshi Nakamura[††] and Kisaburo Nakazawa[†††]

We evaluate the performance massively parallel processor CP-PACS on NAS Parallel Benchmarks Kernel CG. Since Kernel CG requires global data transfer and short vector processing, it is hard to achieve performance in proportion to the number of PU's. However, the pseudo vector processing mechanism implemented in each node processor can effectively handle short vector calculations. Moreover, Hyper-Crossbar Network, the interconnection network of CP-PACS, can provide high-performance global data communication. Owing to these characteristics, CP-PACS can achieve high absolute performance and speed-up ratio when increasing the number of PU's.

## 1. はじめに

超並列計算機CP-PACS[1),2)]は，主に計算物理学の諸問題を対象としているが，そのアーキテクチャは様々な数値処理を考慮し設計されている．単体ノードプロセッサは擬似ベクトル処理機構[3),4)]を採用し，短ベクトルから大規模ベクトル処理まで十分な実効性能が引き出せる．さらに，3次元ハイパ・クロスバ[5),6)]によるノード間結合網は，数千台規模での実装容易性とともに高い柔軟性を持っており，ユーザの要求する様々なデータ転送パターンに対して高いスループットを提供することできる．

本論文では，NAS Parallel Benchmarks（以下NPB）version 1[7)]のうちKernel CGベンチマークによって，超並列計算機CP-PACSの性能評価を行う．NPBは非常に多くの並列計算機において評価がなされており，性能評価の指標として有効である．NPBは，version 2[8)]からMPIによって記述されたベンチマークプログラムでの評価が行われているが，プログラムの実装をユーザに任せたベンチマークの評価もversion 1として引き続き行われている．

Kernel CGでは大規模疎行列とベクトルの積の計算が処理時間の大部分を占め，この時間をいかに短縮するかが高速化のポイントとなる．一般に超並列計算機のプログラム・チューニングでは，いかにデータ転送時間を削減するかが重要である．この点において，隣接転送だけでなく広範囲な転送に対しても高いスループットを提供するCP-PACSは，ユーザに自由な問題マッピングと並列化手法を提供し，データ転送時間を

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, University of Tokyo
††† 明星大学情報学部
Faculty of Computer Science, Meisei University

削減するために様々なアルゴリズムを試みることを可能にする．しかし，このような並列化が内部処理時間に影響を与えることも考慮に入れる必要がある．なぜなら，擬似ベクトル処理機構を持つCP-PACSのノードプロセッサでは，内部処理時間はベクトル長に依存するからである．これからのハイパフォーマンス・コンピューティングの分野においては，ベクトル処理機構を並列計算機に組み込むことは必須といえ，ノードプロセッサでの処理効率を落とさないように，アルゴリズムを検討する必要がある．

本論文では単にベンチマークプログラムの時間計測を行うだけではなく，処理時間をクロックカウンタを用いて細かく分解し，その解析を行う．特に主なデータ転送によって起きる並列処理のオーバヘッドの時間はCP-PACSの諸元から予想し，実測との比較を行うことでプログラマの想定した最適化が実測に反映していることを示す．また，擬似ベクトル処理機構を持つプロセッサの短ベクトル処理に対する特性を明らかにし，その有効性についても考察する．

## 2. 超並列計算機 CP-PACS

超並列計算機CP-PACS[1),2)]（図1）は広い範囲の数値計算処理を効率良く実行できるように，その設計・実装段階において様々なアーキテクチャ上の特徴を採り入れている．現在のCP-PACSは計算用ノード（PU）が2048台，ディスクなどI/O装置を接続するためのノード（IOU）が128台あり，それらは8×17×16の3次元ハイパ・クロスバ結合網[5),6)]（以下HXB）によって接続されている．各PUには64 MBのメインメモリ，16 KB(I)/16 KB(D)の1次キャッシュメモリ，512 KB(I)/512 KB(D)の2次キャッシュメモリがある．

3次元HXBとPUはNIA（Network Interface Adapter）によって接続されており，そのスループットは300 MByte/secである．3次元HXBは図1に示すように3次元直行空間上にPUを並べ，その間をクロスバスイッチ（XB）で結合した間接網である．各次元方向のPUは単一のXBによって完全結合され，それ以外のPU間のデータ転送はエクスチェンジャ（EX）と呼ばれる小型のクロスバスイッチによって複数次元方向のXBを経由して行う．CP-PACSではメッセージ転送にwormholeルーティングを用い，経路の選択には固定ルーティングを用いている．

3次元HXBはmesh/torusネットワークはもちろんHyperCubeネットワークも部分的にエミュレートすることが可能であり，バタフライ転送のような広域

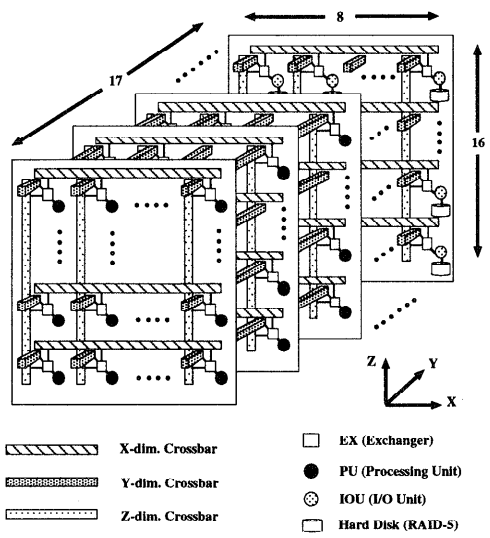


図1 CP-PACSの構成図
Fig. 1 Overview of CP-PACS architecture.

な転送も無衝突で行える[9)]．このため，これらの相互結合網を前提にチューニングされた，従来の並列アルゴリズムをHXBに適用することも可能である．

さらに，CP-PACSにはリモートDMAと呼ばれる高速なPU間データ転送方式がある．これは，PUのNIAがユーザメモリ空間のデータを直接DMAアクセスして送受信を行うものである．特に受信側では受信命令を発行する必要はなく，メッセージ到着によって自動的にNIAが動作し，ゼロ・コピーでユーザメモリ空間にデータが書き込まれる．つまりOS領域のバッファを使用しないので，ハードウェアの高速なスループットをそのまま活かすことができる．また，転送の立ち上げオーバヘッドを軽減するために，NIAの起動をOSを介さずにユーザが直接行えるようにもなっている．

次にPUの特徴について述べる．PUには，既存のRISCプロセッサ（HP PA-RISC 1.1）をベースに擬似ベクトル機構（PVP-SW: Pseudo Vector Processor based on Slide Window）[3),4)]を付加したノードプロセッサを使用している．超並列計算機は非常に多数のノードプロセッサを用いるため，その量的な制約から1チップで構成されるRISCマイクロプロセッサを使用することが妥当である．しかし，既存のRISCプロセッサの性能はキャッシュメモリのヒット率に強く依存している．そのためワーキングセットがキャッシュ容量より大きい問題では，長い主記憶アクセス・レイテンシのために性能は著しく低下する．たとえば，多重ループによって使用されるデータ領域をブロック化

し，アクセスパターンを変更することによって，局所的なワーキングセットを限定し，キャッシュヒット率を向上させる手法も提案されている．しかし，このブロック化の最適なサイズはキャッシュメモリのサイズや構成に大きく影響されるため[10]，注意深い最適化が必要であり，プログラムの可搬性は悪くなる．

これに対しPVP-SWは，キャッシュメモリを介さずに，浮動小数点データをメインメモリから直接レジスタに読み込むpreload命令と，レジスタの内容を直接メインメモリに書き戻すpoststore命令を備える．そして，preload命令とpoststore命令はnon-blocking operationとし，後続命令がload/store命令であってもパイプライン的に処理される．これによって主記憶アクセス・レイテンシを隠蔽する．さらに，このnon-blockingのload/store命令を有効に動作させるために，メインメモリは16バンク構成の擬似パイプライン・メモリを用いる．

具体的には，演算に必要なデータに対するpreload命令を十分前もって先行発行させることで主記憶アクセス・レイテンシを隠蔽する．しかし，preload命令と演算命令の間の距離（両命令の間にスケジュール可能な命令数）はレジスタ数によって制限される．通常のRISCプロセッサのたかだか32本の浮動小数点レジスタでは主記憶アクセス・レイテンシを隠蔽するのは難しい．そこで，CP-PACSのPVP-SWでは浮動小数点レジスタ数を128本に拡張することで長い主記憶アクセス・レイテンシを隠蔽する．また，preload命令とpoststore命令はすべてのレジスタを指定できるが，命令アーキテクチャレベルの上位互換性を保つために，通常の命令はレジスタ・ウィンドウを介して，拡張したレジスタをアクセスする．

擬似ベクトル処理はループ処理におけるベクトル演算を構成する命令を，スーパスカラ方式によって順次並列発行することによって実現される．ループの1イタレーション内では実際の演算と，将来必要とされるデータを先のウィンドウのレジスタへpreloadする処理が並行して行われる．さらに，スーパスカラ方式によってpreload/poststore命令と浮動小数点演算命令の2命令同時発行が可能であり，ベクトル処理を効率的に行う．CPUは動作周波数が150 MHzであり，1命令で浮動小数点のmultiplyとaddを行う複合命令によって理論ピーク性能は300 MFLOP/sである．

現在のCP-PACSは2048台のPUを1024台，512台，256台×2の4パーティションに分割しており，今回の性能測定には256台（4×8×8）のパーティションを使用した．

## 3. NPB Kernel CGの並列化

本章ではNPB version 1 Kernel CG[7]のアルゴリズムを示し，CP-PACSでの並列化ついて述べる．

Kernel CGは正値対称な大規模疎行列の最小固有値をCG（Conjugate Gradient）法によって求めるベンチマークである．Class Bの問題サイズでは疎行列のサイズは75000×75000，非零要素率は0.24%であり1行に平均183個ある．以下に逐次版Kernel CGのメインループを示す．

```
do k=1,KERNITR
   call matvec(a, colidx, rowstr, p, q)
   alpha=rho/dotpro(p, q)
   z(1:N)=z(1:N)+alpha*p(1:N)
   rho0=rho
   r(1:N)=r(1:N)-alpha*q(1:N)
   rho=dotpro(r, r)
   beta=rho/rho0
   p(1:N)=r(1:N)+beta*p(1:N)
enddo
```

ここで，p, q, z, rは$N$（=75000）要素のベクトルであり，matvecとdotproはそれぞれ行列・ベクトル積とベクトルの内積を計算するサブルーチンである．内積計算（dotpro）といわゆるDAXPYの計算は3次元PU空間を仮想的に展開した1次元PU空間に各ベクトルをブロック分割し，データパラレル処理によって行う．これにより，すべてのPUがデータ転送を行うことなしに並列処理を行える．なお，内積計算ではPU内の計算が終了した後にButterfly Summation[11]を行い，全PUが同じ計算結果を保持する．

メインループにおいて最も処理時間のかかる部分は行列・ベクトルの積（matvec）である．matvecの処理は仮想的な2次元PU平面（$P = P_x \times P_y$，$P$はPU台数）に行列をブロック-ブロック分割する．これは，単に仮想1次元PUに対して行ブロックマッピング（$P_x = 1$）や列ブロックマッピング（$P_y = 1$）を行った場合も包含しており，この$P_x$, $P_y$の組合せによりデータ転送時間の最適化が行えると考えられる[12]．

なお，各PUでは部分疎行列を値（a）と桁位置（colidx）と各行のスタートポイント（rowstr）によって表現する．以下にmatvecの並列処理を示す．**図2**は例として8PU（$P_x = 2$, $P_y = 4$）によるmatvecの処理過程におけるデータ転送および演算を示している．図中，斜線部は5番のPUが処理するデータを表している．なお，行列$A$は簡単のため密行列表現されているが，実際には疎行列である．

**(1) Collection：** 被乗数ベクトルpに関し，各PU

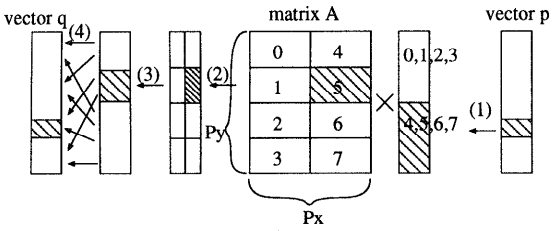


**図2** matvec の処理とデータ転送パターン
Fig. 2 Data processing and transfer pattern in "matvec".

はその保持する部分行列の列に対応する $N/P_x$ 要素を必要とする．しかし，初期状態として p は $N/P$ に分割されているために $P_y$ 台の PU からなる $P_x$ 個の各グループ内で p の部分ベクトルを集め，$N/P_x$ 要素のベクトルを作る必要がある．ここでは，この処理をベクトルの Collection と呼ぶ．ベクトルの Collection の方法としては $P_y$ 回のマルチキャスト，$P_y-1$ 回の「バケツリレー」的な転送などがあるが，広範囲な転送が得意な HXB では各 PU の転送回数が少ない butterfly 転送によって行うのが効率的である．転送パターンは Butterfly Summation と同じであるが，Butterfly Summation がリダクション演算であるために毎回のデータ転送量が一定なのに対して，Butterfly Collection ではデータ転送量が 2 倍ずつ増加する点が異なる．

**(2) Multiply**： 次に部分行列と部分ベクトルの乗算を行う．この乗算結果はベクトル q に保持されるが，q の 1 要素の計算は以下のようなリストベクトル処理になる．

$$\mathtt{q} = \mathtt{q} + \mathtt{a}[i] \times \mathtt{p}[\mathtt{colidx}[i]]$$

浮動小数点データ (a, p) のロードはレジスタへの preload 命令を使うコードがコンパイラによって生成される．これに対して colidx は整数データなので，そのアクセスは通常のロード命令によって行われるが，整数配列の連続領域に対するアクセスなので 1 次キャッシュの高ヒット率が期待できる．

**(3) Summation**： Multiply で求めた結果は部分和なので，真の乗算結果を得るために $P_x$ 台の PU からなる $P_y$ 個の各グループ内で Vector Summation を行う．この処理は dotpro の Butterfly Summation と同等のリダクションをベクトルに対して行う．

**(4) Shuffle**： Summation で求まった部分ベクトル q はこの後で dotpro や DAXPY の計算に用いられるが，各 PU が保持している部分は他のベクトルのマッピングと異なってしまっている．そこで，各 PU が保持する部分ベクトル q の位置を合せるために Shuffle 転送を 1 回行う．

**表1** matvec のデータ転送
Table 1 Data transfer in "matvec".

| 処理パターン | 転送回数 | 総転送量 |
|---|---|---|
| Collection | $\log_2 P_y$ | $N/P \times (P_y - 1)$ |
| Summation | $\log_2 P_x$ | $N/P_x \times \log_2 P_x$ |
| Shuffle | 1 | $N/P$ |

## 4. 実行時間予測

matvec の処理時間は用いる 3 次元 PU 空間をどのような仮想 2 次元 PU 平面（$P_x \times P_y$）に展開するかによって決まる．まず，matvec で行われるデータ転送の回数とデータ転送量を**表 1** に示す．定性的には $P_x$ の増加は Summation 時間の増加を招き，$P_y$ の増加は Collection 時間の増加を招く．簡単なピンポン転送の実験からデータ転送の立ち上げオーバヘッドは約 5500 clock☆，転送スループットは 2 Byte/clock という基本通信性能が分かっているので全転送時間は以下のようになる．

$$(\log_2 P_x + \log_2 P_y + 1) \times 5500 + N/P \times (P_y + P_y \log_2 P_x)/2$$

次に，CPU の計算時間を求める．計算時間は Multiply の時間と Summation のベクトル和の時間からなる．疎行列の非零要素の分布がほぼ一様なので 2 次元 PU 平面の構成に関係なく Multiply の演算数は一定であると考えられるが，部分行列とベクトルの積は行ごとに行われ，その要素数は $P_x$ の増加によって減少する．これは PVP-SW のようなベクトル処理機構にとって不利な条件であり，$P_x$ の増加にともなって演算時間は増加すると考えられる．しかし，ここではこの増加量の見積りが難しいために，十分なベクトル長があるものと判断し，Multiply の演算時間は一定であると予測しておく．

Summation 時のベクトル和の演算量は $N/P_y$ なので $P_x$ の増加 ($= P_y$ の減少) は演算時間の増加を招く．ベクトル和の部分のみを実測した結果，1 要素あたり 7 clock で演算できることが分かったので，Multiply の処理時間を固定とすると Multiply を除く matvec の処理時間は**図 3** のように予想され，PU 台数が 256 と 128 のときは $P_x = 4$，64 以下では $P_x = 2$ のときが最も高速に処理できると考えられる．

☆ プログラミングを容易にするために最速の送信命令は使っていない．

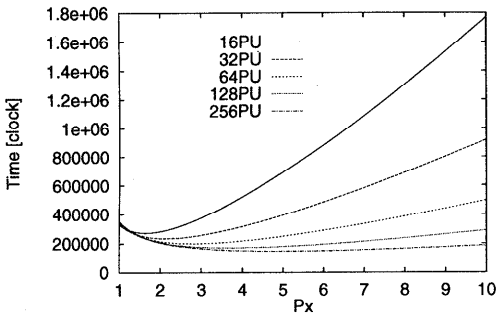


図3 Multiply を除く matvec の予想処理時間
Fig. 3 Estimation time in "matvec" except Multiply.

表2 Kernel CG の処理時間 [sec]
Table 2 Result of Kernel CG [sec].

| #PU | $P_x = 1$ | $P_x = 2$ | $P_x = 4$ | $P_x = 8$ |
|---|---|---|---|---|
| 16 | 106.21 | 126.41 | 203.43 | 252.83 |
| 32 | 55.75 | 66.18 | 105.24 | 129.66 |
| 64 | 30.89 | 35.07 | 55.34 | 69.11 |
| 128 | 19.66 | 20.54 | 30.44 | 39.00 |
| 256 | 13.19 | 13.00 | 17.66 | 19.73 |

表3 メインループの処理時間の内訳 (256 PU) [clock]
Table 3 Breakdown of elapse time for the main loop (256 PU) [clock].

| 処理 | $P_x = 1$ | $P_x = 2$ | $P_x = 4$ | $P_x = 8$ |
|---|---|---|---|---|
| matvec | 804846 | 797683 | 1143634 | 1290212 |
| dotpro | 55478 | 59084 | 61063 | 59976 |
| DAXPY | 3115 | 3130 | 3207 | 3059 |

表4 matvec の処理時間の内訳 (256 PU) [clock]
Table 4 Breakdown of elapse time for "matvec" (256 PU) [clock].

| 処理 | $P_x = 1$ | $P_x = 2$ | $P_x = 4$ | $P_x = 8$ |
|---|---|---|---|---|
| Collection | 444295 | 253017 | 148965 | 86151 |
| Multiply | 351274 | 490649 | 873599 | 1091424 |
| Summation | 94 | 15115 | 40072 | 87841 |
| Shuffle | 9757 | 6914 | 6874 | 6749 |

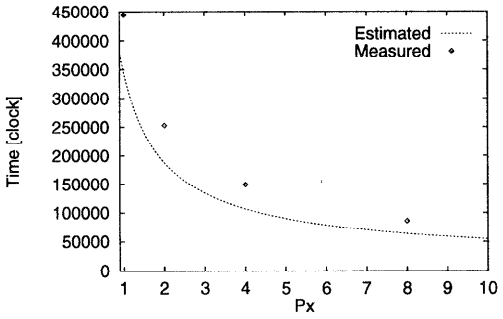


図4 Collection の処理時間 (256 PU)
Fig. 4 Elapse time of "Collection" (256 PU).

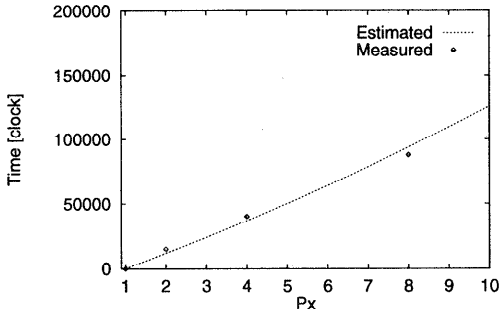


図5 Summation の処理時間 (256 PU)
Fig. 5 Elapse time of "Summation" (256 PU).

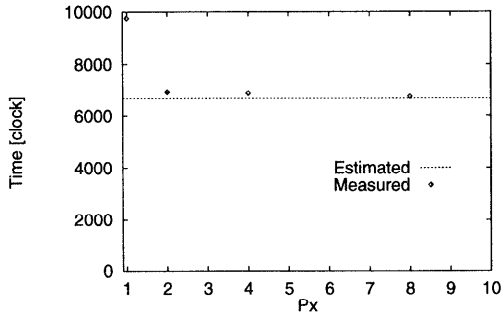


図6 Shuffle の処理時間 (256 PU)
Fig. 6 Elapse time of "Shuffle" (256 PU).

## 5. 性能評価

### 5.1 データ転送時間の最適化

PU 台数が 16 から 256 の場合において，$P_x$ を 1 から 8 まで変えて Kernel CG の実行時間を測定した結果を**表2** に示す．この結果では，ほぼすべての PU 台数で $P_x = 1$ のときが最短の処理時間を示し，予想とは異なる結果となった．この原因を究明するために，PU が 256 台の場合について，メインループの処理時間の内訳と matvec の処理時間の内訳を測定した．これらを**表3**，**表4** に示す．さらに，データ転送パターンごとの予測値と実測値を**図4**，**図5**，**図6** に示す．

これらの測定には，CPU に内蔵されている，clock に同期してインクリメントされる特別なコントロールレジスタ（すなわちクロックカウンタ）を用いた．これにより，ほぼ clock 周期（= 6.7 ns）の精度できわめて正確な処理時間の測定が可能である．

表3 から分かるように，最も処理時間のかかる部分は，予測したとおり matvec である．matvec の処理時間の内訳を表4 から見ると，Multiply の処理時間は $P_x$ にともなって増加していることが分かる．これ

は，前章での予測と大きく異なる点であり，表2においてほぼすべてのPU台数の場合に $P_x = 1$ が最適となってしまっている原因でもある．一方，matvecにおける各転送パターンごとの予測時間と実測時間を比較すると（図4，図5，図6），予測値と実測値が良い精度で合っていることが分かる．つまり，行列のマッピングによってCollectionとSummationの転送時間がトレードオフしている．ここで，Shuffle転送に関しては多次元のHXBネットワークを用いた場合にネットワーク中で衝突が起こることが分かっているが，予測値は無衝突を仮定しているので若干の誤差が生じている．しかし，Shuffle転送は他の転送に比べて転送回数や転送時間が小さいために，この誤差は全体の予測には影響を及ぼさないといえる．

表4において，Multiplyの処理時間が予想に反して $P_x$ にともなって増加している点に関しては次節で述べる．

### 5.2 CPU処理時間の最適化

Multiplyの処理時間はPU内での純粋な計算時間のみであり，「演算量が一定なので計算時間が2次元PU構成によらず一定である」と仮定したが，CP-PACSのPUはPVP-SW機構によりベクトルプロセッサと同じような特性を持ち，ループ長が短いループ処理では性能が低下する．コンパイラの生成する擬似ベクトル処理のコードは，一定の長さのベクトル長（CVL: Critical Vector Length）を必要とし，実際のベクトル長がCVL以下であればスカラ処理，CVL以上であれば擬似ベクトル処理を行うようなコードが生成されている．

現在のコンパイラでは，メモリのバンク・コンフリクトなどによる主記憶アクセス・レイテンシの増加に対しても可能な限り対応するために，使用可能な浮動小数点レジスタ（約100本程度）を最大限に利用し，preload命令と実際の演算命令の間に他のデータのpreloadを挿入することによって，主記憶アクセス・レイテンシを極力隠蔽するような擬似ベクトルコードを生成する．このために，一般にCVLは大きくなりMultiplyのコードではCVL=50であった．これに対して，今回の測定に用いたプログラムでは部分行列の1行ごとに積を計算するので，ループ長は1行の非零要素数になる．Kernel CG（Class B）では1行の平均非零要素数は183個であり，平均ベクトル長は $183/P_x$ となる．つまり，コンパイラの生成したCVL = 50のコードでは，$P_x$ の増加にともなってベクトル長が短くなり，通常のload命令によるスカラコードでMultiplyの処理が行われる場合が増加するためにMultiplyの処理時間が増加してしまう．

CVL = 50では，実際の演算が始まるまでに50要素分のpreloadを必要とし，この時間はベクトル処理でのスタートアップ時間に相当する．しかし，PVP-SWはスーパスカラ処理のプロセッサにデータ供給能力を強化して擬似的にベクトル処理を行うので，このスタートアップ時間を短くしたコードを作成することも可能であり，本質的には短ベクトル処理に対応できる．そこで，我々はMultiplyの部分においてのみアセンブラソースレベルのチューニングを行い，CVL = 20のプログラムを作成して評価を行った．このようなアセンブラレベルのチューニングはNPBのルールでは禁止されているが，コンパイラへのディレクティブなどを用いてCVLをFORTRANソースレベルでチューニングすることは容易であり，CP-PACSの本質的な性能評価を行う目的で有効であると考える．

**表5**はCVL = 20のアセンブラコードによるMultiplyの時間を示している．$P_x = 1$ のときはほとんどのベクトル長が50を超えているためにMultiplyの処理時間はCVL = 50のコード（表4）とほぼ同じであるが，$P_x = 2$ では35%，$P_x = 4$ では39%短縮される．これは，$P_x$ が大きくなるのにともなって短ベクトルの割合が増えるためである．$P_x = 8$ では29%の短縮にとどまっているが，これはCVL = 20のコードでもスカラ処理になってしまう20以下のベクトル長の割合が増えるためである．このMultiplyを用いた全体の処理時間を**表6**，**図7**に示す．この結果は予想転送時間（図3）に単調増加するMultiplyの処理時間を足した振舞いをし，PU台数が大きいときには $P_x$ にともなうMultiplyの時間増加分が通信時間の変化とほぼ同じオーダになるので，$P_x$ によって処理時間がトレードオフする．

以上のように，通信時間の最適化とPU内部処理時

**表5** matvecの処理時間の内訳（256 PU, CVL = 20）[clock]
Table 5 Breakdown of elapse time for "matvec" (256 PU, CVL = 20) [clock].

| 処理 | $P_x = 1$ | $P_x = 2$ | $P_x = 4$ | $P_x = 8$ |
|---|---|---|---|---|
| Multiply | 357818 | 419084 | 531332 | 769355 |

**表6** Kernel CGの処理時間（256 PU, CVL = 20）[sec]
Table 6 Result of Kernel CG (256 PU, CVL = 20) [sec].

| #PU | $P_x = 1$ | $P_x = 2$ | $P_x = 4$ | $P_x = 8$ |
|---|---|---|---|---|
| 16 | 101.25 | 110.22 | 138.96 | 198.46 |
| 32 | 54.51 | 58.98 | 71.04 | 100.95 |
| 64 | 30.30 | 31.09 | 36.93 | 52.54 |
| 128 | 18.38 | 17.59 | 19.92 | 27.90 |
| 256 | 12.94 | 11.38 | 11.86 | 15.41 |

表7 他の並列計算機のKernel CGの処理時間 [sec]
Table 7 Results of Kernel CG on other machines [sec].

| #PU | | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|---|
| CP-PACS | | 101.25 | 54.51 | 30.30 | 17.59 | 11.38 |
| CRAY T3E | (Nov 96) | 107.1 | 59.3 | 33.8 | 22.9 | 22.2 |
| IBM RS/6000 SP Wide-node 1 (67 MHz) | (Mar 94) | 88.4 | 52.53 | 33.79 | 25.44 | – |
| IBM RS/6000 SP P2SC node (120 MHz) | (Nov 96) | 62.21 | 36.22 | 22.71 | – | – |

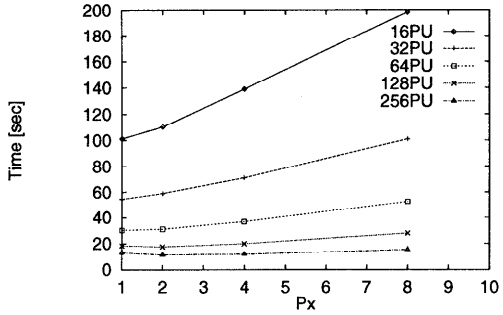


図7 Kernel CGの処理時間 (CVL = 20)
Fig. 7 Result of Kernel CG (CVL = 20).

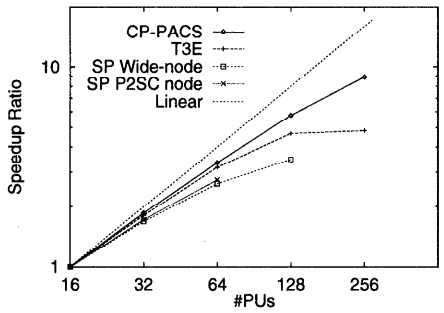


図8 速度向上率
Fig. 8 Speed up ratio.

間の最適化がトレードオフする場合，ここに示したような注意深いチューニングが必要であることが分かった．この種の問題は，ハイパフォーマンス・コンピューティングにおいて大きな位置を占める，ベクトル処理機能を持つPUを用いた並列計算機において，今後重要になっていくものと思われる．

## 6. 他の並列計算機との比較

最後にNPB version 1の結果[13]を基に，Cray T3E, IBM RS/6000 SP system と CP-PACS を比較する．各並列計算機の結果を**表7**に，各々の16 PUの処理時間を1とした速度向上率を**図8**に示す．

Cray T3Eと比較するとすべてのPU台数において処理時間が短く，特にPU台数が増えたときにCP-PACSは台数効果を維持していることが分かる．また，RS/6000 SP P2SCに対しては，ノードプロセッサのpeak性能差（CP-PACS: 300 MFLOP/s, SP P2SC: 480 MFLOP/s）から絶対時間はCP-PACSが劣る．しかし，SP P2SCでは64 PUの段階で性能が頭打ちになり始めているのに対して，CP-PACSは台数効果を維持していることが分かる．T3EやSP P2SCでの測定に用いられた並列アルゴリズムは公開されていないが，PU台数の増加によってデータ転送の粒度が細かくなることが予想される．こうような細粒度のデータ転送に対して，CP-PACSのリモートDMAによる低オーバヘッドの通信が有効に働き，CP-PACSでは台数効果が維持できているといえる．

## 7. おわりに

本論文では，NPB kernel CG（Class B）によって超並列計算機CP-PACSの性能評価およびその解析を行った．CPUに組み込まれているいわゆるクロックカウンタを用いることにより，様々な処理時間を細かく測定・解析することができた．解析の結果，データ転送に関しては，基本的な諸元（転送立ち上げオーバヘッドとスループット）から予測される時間と実測がほぼ一致することを確認した．このことから，多くの転送パターンを無衝突で扱えるHXBではデータ転送時間の正確な見積りが可能であることが分かった．

Kernel CGは様々なデータ転送パターンを必要とし，その多くが細かいデータ転送となるために並列化効率が悪い．この問題を解決するために行列を2次元PU平面にブロック-ブロック分割しデータ転送を最適化する手法を用いた．しかし，このマッピングがPVP-SWによる擬似ベクトル処理のベクトル長を短くしてしまい，内部処理時間の増大を招く結果となった．現在のコンパイラはバンクコンフリクトなどによる長い主記憶アクセス・レイテンシにできるだけ対処するようなコードを作成するが，逆にそのようなコードが擬似ベクトル処理をする最短ベクトル長を大きく

してしまう．今回の評価対象であるKernel CGのような短ベクトル処理を多く行う問題では擬似ベクトル処理機構を有効に活用できないことが分かった．これに対して，擬似ベクトル処理の最短ベクトル長（CVL）を小さくすると，短ベクトル処理においても擬似ベクトル処理が効果的であることが分かった．

一般に超並列計算機では通信時間を減らすためにアルゴリズムの選択や細かいチューニングが行われるが，これに対してHXBは広い選択範囲を与える．さらにハイパフォーマンス・コンピューティングの分野ではベクトル処理は必須であり，この性能は処理のベクトル長に依存する．これに対しては短ベクトルに対してもPVP-SWが小さいオーバヘッドで処理可能であることを示した．この種のチューニングの問題は，今後の並列ベクトル機において重要になっていくと考えられる．

結果として擬似ベクトル処理を最適化することで，他の並列計算機と比較してCP-PACSはKernel CG（Class B）問題に対して高い絶対処理性能と台数効果を持つことを示した．今後は，他のNPB Kernelやアプリケーションに対する評価を行い，CP-PACSの有効性を示したい．

**謝辞** 超並列計算機CP-PACSを利用する機会を与えていただいた筑波大学計算物理学研究センター関係各位に感謝します．なお，本研究の一部は創成的基礎研究費（08NP0401）の補助によるものである．

## 参考文献

1) 中澤喜三郎ほか：CP-PACSのアーキテクチャの概要，情報処理学会研究報告，ARC-108-9 (1994).
2) Iwasaki, Y.: Status of the CP-PACS Project, *Proc. Lattice '96, Nucl. Phys. B (Proc. Suppl.)* (1996).
3) Nakamura, H., et al.: A Scalar Architecture for Pseudo Vector Processing based on Slide-Windowed Registers, *Proc. ACM International Conference on Supercomputing '93*, pp.298–307 (July 1993).
4) 位守弘充ほか：スライドウィンドウ方式による擬似ベクトルプロセッサ，情報処理学会論文誌，Vol.34, No.12, pp.2612–2613 (1993).
5) 田中輝雄ほか：識別子を用いたデータ転送方式を基本とするMIMD型並列計算機アーキテクチャ，並列処理シンポジウムJSPP '89 (1989).
6) 朴 泰祐ほか：ハイパクロスバ・ネットワークの性能評価，信学技報，CPSY-93-40 (1993).
7) Bailey, D., et al.: The NAS Parallel Benchmarks, Technical Report, RNR-94-007, NAS (1994).
8) Bailey, D., et al.: The NAS Parallel Benchmarks 2.0, Technical Report, NAS-95-020, NAS (1995).
9) 松原正純ほか：超並列計算機CP-PACSのネットワーク性能評価，情報処理学会研究報告，HPC-67-10, pp.55–60 (1997).
10) Lam, M., et al.: The cache performance and optimization of blocked algorithms, *Proc. ASPLOS-IV* (Apr. 1991).
11) 板倉憲一ほか：並列計算機CP-PACSのCGベンチマークによる性能評価，情報処理学会研究報告，HPC-63-6, pp.31–36 (1996).
12) Agarwal, R.C., et al.: High-performance parallel implementations of the NAS kernel benchmarks on the IBM SP2, *IBM Systems Journal*, Vol.34, No.2, pp.263–272 (1995).
13) Saim, S., et al.: Parallel Benchmark (Version 1.0) Results 11-96, Technical Report, NAS-96-018, NAS (1996).

(平成 9 年 11 月 7 日受付)
(平成 10 年 4 月 3 日採録)

**板倉 憲一**（学生会員）
昭和44年生．平成5年筑波大学第三学群情報学類卒業．平成7年同大学院工学研究科前期博士課程修了．現在同研究科後期博士課程に在籍中．並列計算機アーキテクチャの研究に従事し，性能評価方法等に興味を持つ．

**松原 正純**（学生会員）
昭和48年生．平成8年筑波大学第三学群情報学類卒業．平成10年同大学院工学研究科前期博士課程修了．現在同研究科後期博士課程に在籍中．並列計算機の研究に従事．

**朴 泰祐**（正会員）
昭和59年慶應義塾大学工学部電気工学科卒業．平成2年同大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和63年同大学理工学部物理学科助手．平成4年筑波大学電子・情報工学系講師，平成7年同助教授，現在に至る．超並列計算機のアーキテクチャおよび性能評価，ハイパフォーマンス・コンピューティングへの応用の研究に従事．電子情報通信学会会員．

**中村　宏**（正会員）

昭和60年東京大学工学部電子工学科卒業．平成2年同大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，平成8年より東京大学先端科学技術研究センター助教授．計算機アーキテクチャ，ハイパフォーマンスコンピューティング，計算機の上位レベル設計支援の研究に従事．本会平成5年度論文賞，平成6年度山下記念研究賞各受賞．電子情報通信学会，IEEE，ACM各会員．

**中澤喜三郎**（正会員）

昭和30年東京大学工学部応用物理卒業．昭和35年同大学院数物系博士課程応用物理修了．同年日立製作所入社．TAC，HITAC 5020，E/F，8800/8700，M-200H/280H，680H，S-810等，超大型コンピュータ・スーパコンピュータの開発に従事．平成元年より筑波大学電子・情報工学系教授，計算物理学センター向きの超並列処理システムCP-PACSの研究に従事．平成8年より電気通信大学情報工学科教授，平成10年より明星大学情報学部教授．工学博士．電子情報処理学会，IEEE，ACM各会員．平成5年度本学会論文賞，平成8年度本学会功績賞各受賞．