CB-004

省レジスタアーキテクチャ向け ソフトウェアパイプライニングの評価

Evaluation of Software Pipelining for Architecture with Small Number of Registers

千葉 修一†1 Shuichi Chiba　青木 正樹†1 Masaki Aoki　鎌塚 俊†1 Shun Kamatsuka　松井 雅人†2 Masato Matsui　八代 尚†3 Hisashi Yashiro

1. はじめに

2012 年 9 月から共用開始となったスーパーコンピュータ「京」（以降，「京」と略す）[1]は，TOP500，ゴードンベル賞など様々なタイトルを獲得している．2018 年 4 月現在でも大規模グラフ解析の性能ランキングとなる Graph500，および共役勾配法の処理速度の性能ランキングとなる HPCG（High Performance Conjugate Gradient）では，世界ランキング 1 位を有している．これらの成果は，CPU やネットワークを中心としたハードウェアだけでなく，コンパイラなどのソフトウェアも含めたシステム全体のバランスを考えた設計によるものである．また，計算科学分野のみならず産業利用などの幅広い分野のアプリケーションで利用されている．「京」の CPU を中心とした諸元をTable 1に示す．

Table 1 K computer System Specification

CPU Specification	
Peak performance	128GFLOPS (16GFLOPS×8cores)
CPU clock	2.0 GHz
Floating-point registers (Core spec)	SIMD register (128 bit SIMD) : 128 General purpose register (64bit) : 188
Cache	L1I$/L1D$: 32KiB (2way), L2$: Shared 6MiB (12way)
Power	58W (30℃, Water Cooling)
Network Specification	
トポロジ	6D Mesh/Torus
リンクバンド幅	5GB/s (6.25Gbps x 8 lanes x 10 dirs)
ノードバンド幅	20 GB/s x in/out
Software Specification	
Compiler optimization	HPC-ACE，Reciprocal approximation, HW/SW prefetch, Hardware barrier

「京」のシステムを支える特徴的な技術として，HPC-ACE（High Performance Computing Arithmetic Computational Extension）[2]が挙げられる．HPC-ACE は，SPARC アーキテクチャに対して，HPC（High Performance Computing）向けに拡張を加えたアーキテクチャである．128 本に拡張された SIMD レジスタ(スカラの浮動小数点レジスタとしては 256 本)と，それらを活用する拡張命令（SIMD 命令，逆数近似命令など）を利用することでアプリケーションを高速かつ高効率に実行することを実現している．アプリケーションがこれらの機能を利用するためには，アプリケーションの翻訳時にコンパイラが適用する最適化技術が必要不可欠となる．特に「京」向けに開発されたソフトウェアパイプライニングは HPC 向けのアプリケーションに対して重要である．

ソフトウェアパイプライニングは，命令レベルで並列性を高める最適化技術である．ループ構造の変形，およびループ内の命令列を CPU の演算器が効率的に処理できるように並び替えることで IPC（Instruction Per Cycle）を向上させる．

EuroBen benchmark[3]に含まれる 9 次の多項式を解くコード（Figure 1，以降，9th degree code とする）を例にとり，「京」におけるソフトウェアパイプライニングの効果を評価した．

```
  Do i = 1, n
    y(i,jsw) = c0 + x1(i)*(c1 + x1(i)*
&              (c2 + x1(i)*(c3 + x1(i)*
&              (c4 + x1(i)*(c5 + x1(i)*
&              (c6 + x1(i)*(c7 + x1(i)*
&              (c8 + x1(i)*c9)))))))
  End do
```

Figure 1 9th degree polynomial expression

9th degree code は，回転ごとに配列 x1 のデータを読み込み，9 つの積和演算を行う演算主体のコードである．このコードを用いて評価を行った結果をFigure 2に示す．

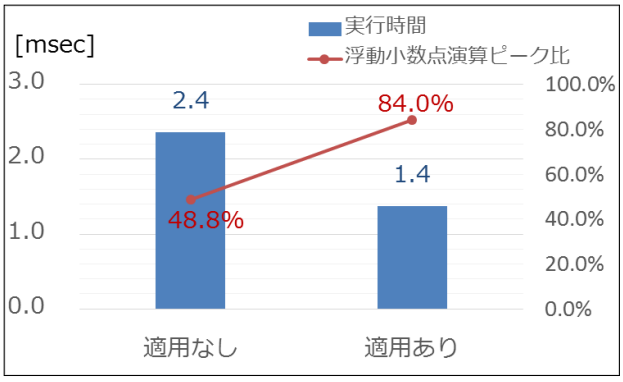

Figure 2 EuroBen benchmark Kernel 14

グラフは，ソフトウェアパイプライニングの適用有無による性能の変化を示している．縦棒グラフは実行時間

†1 富士通株式会社 FUJITSU LIMITED
†2 株式会社メトロ Metro, Inc.
†3 国立研究開発法人理化学研究所 RIKEN

を意味しており，低いほど高速に実行できていることが分かる．ソフトウェアパイプライニングの適用により 1.7 倍の性能向上が見られる．横線グラフは，浮動小数点演算の効率（以降，実行効率と呼ぶ）を示しており，ピークの演算性能に対してどのくらいの演算性能が得られているかの比率を示している．こちらのグラフは，高いほど高効率だと言える．ソフトウェアパイプライニングの適用により，演算数に変化はないが実行時間が短くなったため，実行効率が高くなっていることが分かる．

このように「京」においてソフトウェアパイプライニングは非常に大きな性能効果が得られる最適化であることが分かる．「京」は 2019 年度中に運用終了し次期のシステムとなるポスト「京」[4]へ移行されることとなっている．ポスト「京」ではコデザインを導入することで，システムとアプリケーションがお互いの特性を設計段階で取り込んでいる．これにより，「京」以上の実行効率，および電力効率を実現するシステムの開発が進められている．また，その一環で CPU に Arm プロセッサが採用された．これに合わせて，命令セットアーキテクチャは「京」で利用していた SPARC アーキテクチャから，Armv8-A アーキテクチャへ変更される．さらに HPC 向けに SVE（Scalable Vector Extension）[5]と呼ばれる拡張機能が追加される．SVE では，32 本の SIMD レジスタ(ベクトルレジスタ)，16 本のプレディケートレジスタが利用される．「京」で搭載された SPARC 向けの拡張命令である HPC-ACE との比較をTable 2に示す．

Table 2 HPC-ACE and SVE Specification

	HPC-ACE	SVE
SIMD registers	128	32
General purpose registers	188	31
Predicate registers	-	16

SVE は HPC-ACE に比べ大幅にレジスタ数が少ないことが分かる．そこでソフトウェアパイプライニングを SIMD レジスタ数の観点を中心として評価し，「京」からポスト「京」への技術継承が可否を考察し，ポスト「京」の開発に向けた提案を行う．

2. 評価に利用する環境

ソフトウェアパイプライニングを評価するために利用する CPU の概要について述べる．ポスト「京」は現在開発中であるため，以下の 2 つの観点で評価を行える環境を用意した．

① SIMD レジスタ数の変化（128 本，32 本）

② 命令セットアーキテクチャの変化（SPARC，Arm）

評価①については，「京」の後継機である富士通製 PRIMEHPC FX100（以降，FX100 と呼ぶ）を利用した．また，FX100 向けのコンパイラを改造し，プログラムが利用する最大の SIMD レジスタ数を変更する機能を追加した．この機能を利用することで 32 レジスタの評価を実現した．評価②については，評価①で利用する FX100 と Arm チップである X-Gene 2[6]が搭載されたマシンを比較することで評価を行った．

富士通製 FX100 の諸元をTable 3に，MACOM 社製の X-Gene 2 が搭載されたサーバの諸元をTable 4に示す．

Table 3 SPARC64™ XIfx Specification

	Specification
Peak performance	1TFLOPS 以上(倍精度)
CPU clock	2.2 GHz
Core	32 +2 (*)
SIMD length	256 bit
Registers (Core spec)	SIMD register (256 bit wide SIMD) : 128 General purpose register (64bit) : 188
Cache	L1I$/L1D$: separate 64KiB (4way) L2$: shared 24MiB (24way)
Memory throughput	240GiB/s x2 (R/W)

(*) 2 個はアシスタントコア

Table 4 X-Gene 2 Specification

	Specification
Peak performance	Unknown
CPU clock	2.8 GHz
Core	8
SIMD length	128 bit
Registers (Core spec)	SIMD register (128 bit wide SIMD) : 32 General purpose register (64bit) : 31
Cache	L1I$/L1D$: Separate L2$: Shared L3$: Globally shared 8MB
Memory throughput	Unknown

コンパイラは，富士通社製コンパイラ（Technical Computing Suite V20L30 相当）を利用した．ただし，X-Gene 2 に関しては，FX100 向けのコンパイラを改造し，コード出力を Armv8-A の AArch64(A64 命令セット)に対応させたものを利用した．

3. ソフトウェアパイプライニングの動作

3.1 ソフトウェアパイプライニングの概要

評価に先立ち，「京」におけるソフトウェアパイプライニングの動作概要について説明する．ソフトウェアパイプライニングは，ループの命令列の実行順序を並べ替えることで，命令レベル並列性を向上させる最適化である．ループのイテレーションを跨いで命令の移動を行い，ループを再構成する形で，実行性能を向上させる．

「京」ではソフトウェアパイプライニングのフレームワークとしてモジュロスケジューリング[7] [8] を採用している．「京」に実装されたモジュロスケジューリングは，大きく分けて二つの処理からなる．一つは，命令レベル並列性を高くするために，各サイクルの演算器資源の利用効率をあげる視点で命令をスケジューリングする．もう一つは，スケジューリング結果を適切にループに反映するために，レジスタリネーミングを施しながら，ループを複製する．これはモジュロリネーミングとよばれる．

ソフトウェアパイプライニング適用後のループにおいて，元ループの各イテレーションの命令が，何サイクル数ずつずれて始まるかを表す値を II (Initiation Interval)とよぶ．これは最適化前のループ一回転分の命令を処理できるサイクル数に相当し，小さいほど実行性能が良いという指標となる． Figure 3に仮想的なマイクロアーキテクチャにおけるソフトウェアパイプライニングの結果を示す．

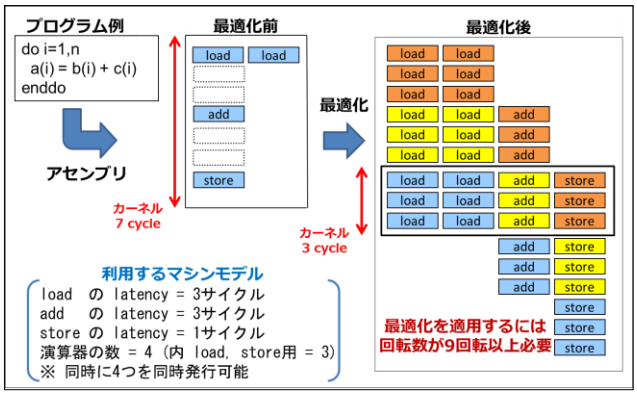

Figure 3 ソフトウェアパイプライニングの動き

ソフトウェアパイプライニングは，IPC の向上効果が得られるようになる一方で，注意を要する点がある．一つは，レジスタリネーミングを実施するため，アーキテクチャレジスタ数を要することである．もう一つは，アンローリングするようにループを複製するため，ループカーネルを通過するために必要なループ回転数を要することである．そのため，ソフトウェアパイプライニングを適用する際は，レジスタやループ回転数の制約のもとで，マイクロアーキテクチャの利用効率を向上させることになる．プログラムの特性としてループの回転数が翻訳時に判明している場合は，その命令配置自体を調整して，アンローリング数を抑えることが可能である．また，レジスタリネーミングによってレジスタが不足することが予想される場合は，命令スケジューリングによる重なりを抑えることで，レジスタのライブレンジの干渉を抑えて必要なレジスタ数を低減させることが可能である．「京」では演算器資源から定まる最小の II を基準として，必要なレジスタ数およびループの回転数を確認しながら，ソフトウェアパイプライニングを実施する．Figure 4に示す処理の流れで適用の有無を判定する．

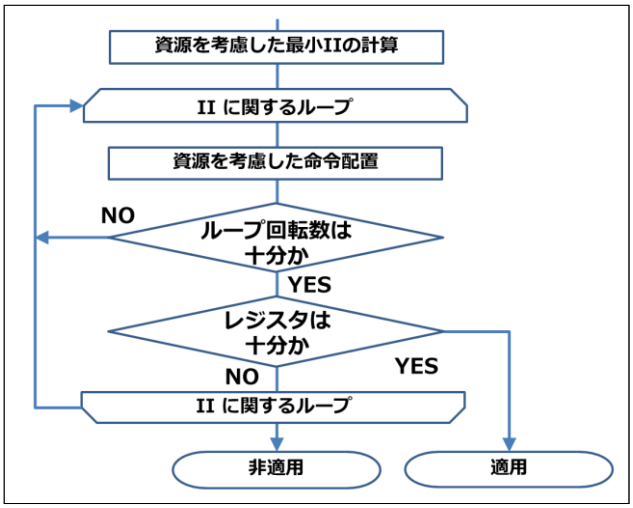

Figure 4 処理の流れ

ここまでで述べた通り，ソフトウェアパイプライニングを実施すると，命令レベル並列性の向上を実現する代わりに，アーキテクチャレジスタが多く必要となる．必要なレジスタ数は，ループ内の命令列やマイクロアーキテクチャの特性，ならびに命令スケジューリング結果に依存するため一概には言えないが，必要なレジスタ数を抑えた命令スケジューリングを実施するアルゴリズムも研究[9]されている．また，ソフトウェアパイプライニングはハードウェアの演算器資源を活用するために，Out-of-order 機構の補助として，パイプラインのストールを削減するための命令スケジューリング最適化である．そのため，例えば，メモリバンド幅がボトルネックとなり実行速度が律速されるループには性能向上効果が期待できない．

3.2 評価環境の違いによる性能の変化

ソフトウェアパイプライニングの効果を評価するために“2. 評価に利用する環境”に示した評価環境で実行時間を測定する．評価で示す実行時間の縦棒グラフは，ソフトウェアパイプライニングの適用をしていないケースを“1.0”とし，適用したケースの実行時間を比率で表現したグラフ（以降，性能効果比グラフと呼ぶ）を用いる．富士通製 FX100 では詳細プロファイラ（fapp）を利用することで，実行時間内の CPU の動作を知ることが可能である．このプロファイラの結果を用いて実行時間内の内訳を表現する．

FX100 の SIMD レジスタを 128 本利用した通常の動作モード（以降，128 レジスタモードと呼ぶ）で，ソフトウェアパイプライニングを適用する場合としない場合のそれぞれについて 9th degree code を測定した結果をFigure 5に示す．

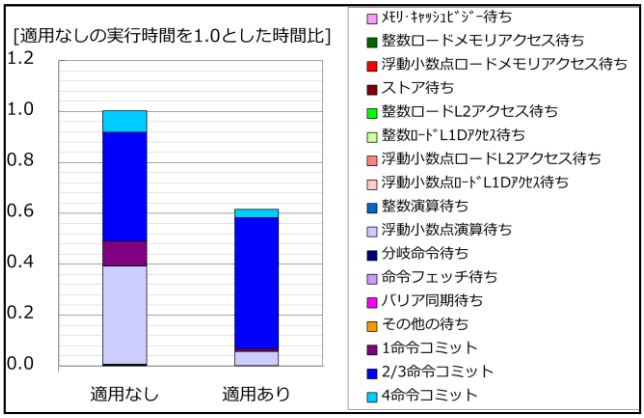

Figure 5 FX100（128 レジスタモード）の評価

ソフトウェアパイプライニング適用なしのケースに対して，適用したケースでは“浮動小数点演算待ち”が削減されていることが分かる．これはソフトウェアパイプライニングによって命令の並列性が促進され，先行する演算の結果を利用して計算する演算処理が待ち合わせなく動作できているためである．

次に FX100 の SIMD レジスタの利用数を 32 本に制限し動作するモード（以降，32 レジスタモードと呼ぶ）で 9th degree code を測定した結果をFigure 6に示す．

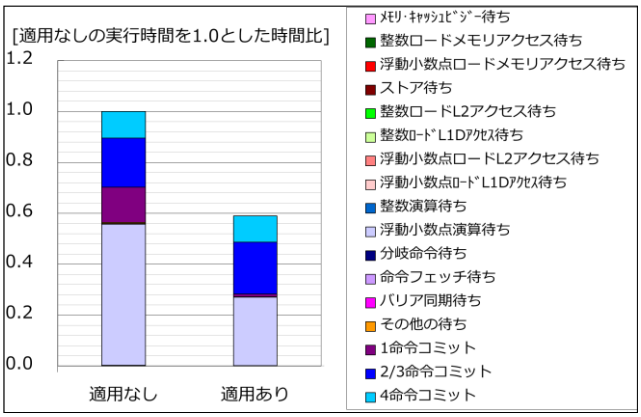

Figure 6 FX100（32 レジスタモード）の評価

32 レジスタモードにおいてもソフトウェアパイプライニングの適用によって性能向上がみられる．その内訳に着目すると，Figure 5とは CPU の動作状況が異なることがわかる．32 レジスタモードでは，ソフトウェアパイプライニング適用後であっても“浮動小数点演算待ち”の割合が多い．そこで，コンパイラが翻訳時に出力する情報からレジスタ数による状況の違いを生む原因を考察する．128 レジスタモードと 32 レジスタモードの適用情報をTable 5に示す．

Table 5 ソフトウェアパイプライニングの適用情報

	128 レジスタモード	32 レジスタモード
ソフトウェアパイプライニングの適用前の最適化		
ループアンローリング	6 展開	3 展開
SIMD 幅	4	4
ソフトウェアパイプライニングの情報		
II	15	22

二つのレジスタモード間では，ソフトウェアパイプライニングの入力となるループアンローリングの適用状況が異なり，32 レジスタモードでは半分の展開数となっている．これは利用できるレジスタ数が少なくなることで，ループボディの命令列も制限されるためである．また，ソフトウェアパイプライニングの動作に関しても，利用できるレジスタの違いに起因して，II の値が異なる．II は値が小さいほど各イテレーションの命令列の重なりが強くなり，より高い IPC を実現し高速に実行できるようになる．先に述べたループアンローリングからの入力の影響も含め，32 レジスタモードでは，II の値が高くなっている．そのため，浮動小数点演算命令の実行完了待ち時間（命令レイテンシ）が隠蔽できず，浮動小数点演算待ちが残っている．これらを総合すると，32 レジスタモードにおいてソフトウェアパイプライニングによる効果は観測できたが，省レジスタにおける課題があるといえる．

次に Arm アーキテクチャの CPU で評価を行うため，MACOM 社製の X-Gene 2 を利用して，ソフトウェアパイプライニングの適用の有無それぞれについて，9th degree code を測定した結果をFigure 7に示す．

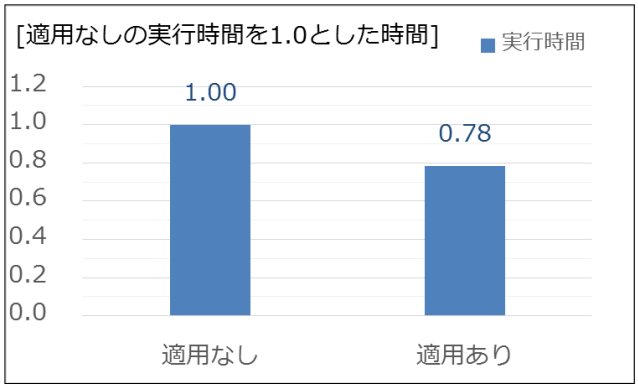

Figure 7 X-Gene 2 を用いた評価

結果より，Armv8-A アーキテクチャである CPU でもソフトウェアパイプライニングの効果が観測できている．先に述べた通り，本システム向けのコンパイラは，FX100 向け（SPARC 向け）のコンパイラをベースとして AArch64 のコード出力に対応したものを利用している．ただし，コンパイラが保持する他の最適化機能にハードウェアやマイクロアーキテクチャ向けのチューニングは実施していない．しかしながら，ソフトウェアパイプライニングは，十分に効果が期待できることが分かった．

これまでの結果から，ソフトウェアパイプライニングの効果を決定する要因として SIMD レジスタ数が想定される．そこで，SIMD レジスタ数に着眼した評価を行った．Figure 8は，FX100 を利用し SIMD レジスタ数を 8 から 128 の間で変化させた場合の 9th degree code の実行性能の比較である．

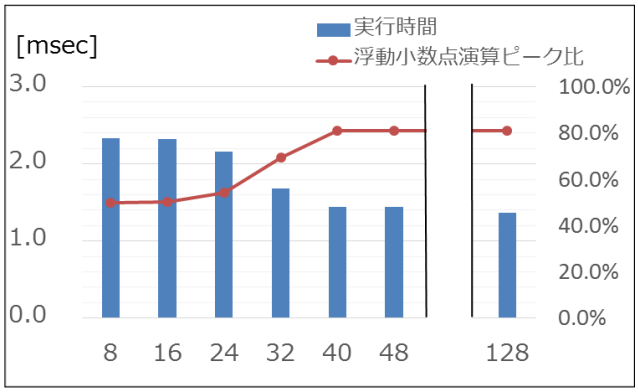

Figure 8 SIMD レジスタ数と実行性能の関係

図より，レジスタ数が多い方がより実行時間が短くなり，性能が向上していることが分かる．ただし，性能の向上は 40 本付近で頭打ちになっている．9th degree code は，SIMD レジスタ数が 40 本あればソフトウェアパイプライニングの十分な効果が引き出せると言える．

4. ベンチマークによる検証

ここまで，9th degree code を中心にソフトウェアパイプライニングの効果を評価してきた．本節からは，一般的なベンチマーク，および実アプリケーションにおける評価を進める．まずベンチマークについて，並列コンピュータ向けの著名なベンチマークである NAS Parallel Benchmark class A（以降，NPB と略す）[10]を利用して評価を行った．NPB は多種のベンチマークを有しているため，128 レジスタモードでソフトウェアパイプライニングの効果をまず検証し，その中で効果が大きいものを対象としさらに検証を進めた．128 レジスタモードで測定した性能効果比グラフをFigure 9に示す．

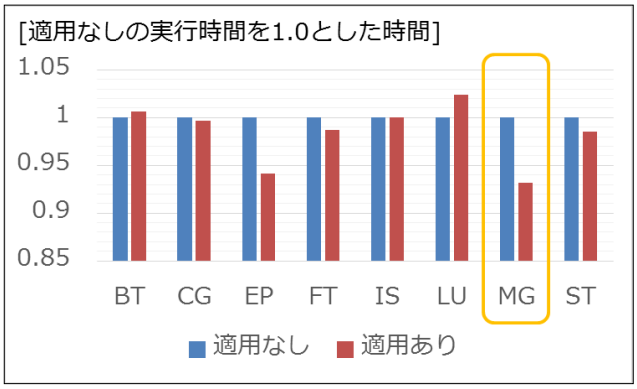

Figure 9 NPB におけるソフトウェアパイプライニングの効果

測定の結果，MG と EP は 6%以上の性能向上が得られた．一方で LU は 2.3%の低下となり，他の 5 本の性能変化は 1.3%以内に収まる結果を得た．ソフトウェアパイプライニングは常に効果が得られるものではなく，アプリによって差があることがわかる．

本節では，ソフトウェアパイプライニングの性能向上を検証するため，向上率が最も大きい MG ベンチマークに着目する．またレジスタ数の影響を確認するため， 32 レジスタモードにおける性能効果を検証した．

4.1 ベンチマークの概要

MG ベンチマークは，三次元ポアソン方程式をマルチグリッド法で解くプログラムとなっている．コードとしては再帰呼び出しにより階層ごとに局所的なデータを用いて演算を行う特徴がある．128 レジスタモードで測定した実行時の処理区間のコスト分布はTable 6のようになっている．またループ部は関数の先頭からの登場順にナンバリングしている．

Table 6 MG ベンチマークのコスト分布

関数 / ループ部	実行コスト割合 (%)
resid / 2nd loop	41.1765
zran3 / 3rd loop	23.5294
resid / 1st loop	11.7647

残差計算の処理である関数 resid の実行コストが合計で 50%以上の割合を占めていることが分かる．また，その中でも二番目のループが 41%の実行コストとなっている．

高コストの 2nd loop ループは，三重ループ構造（制御変数は内側から，i1，i2，i3）となっており，内側の演算はFigure 10のようになっている．

```
  r(i1,i2,i3) =
>   v(i1,i2,i3)
>   - a(0) * u(i1,i2,i3)
>   - a(2) * ( u2(i1) + u1(i1-1) + u1(i1+1) )
>   - a(3) * ( u2(i1-1) + u2(i1+1) )
```

Figure 10 resid 2nd loop のループボディ

配列 a と積算するための配列 u，u1，u2 の加算処理に若干の依存があるが，それ以外の計算は並行で処理可能なものが多い．Figure 10のコードにおける演算の依存関係をFigure 11に示す．

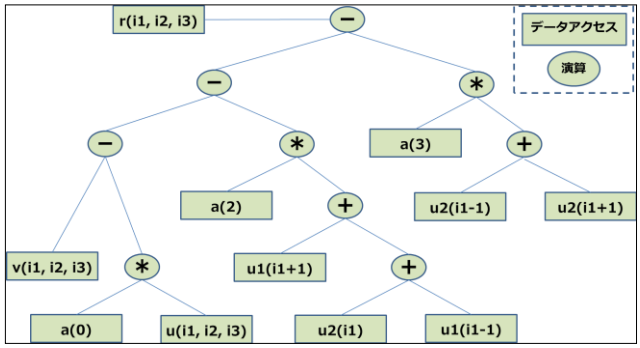

Figure 11　演算の依存関係

配列 a と配列 u，u1，u2 の積算結果を用いた減算計算を頂点に大きく 3 つの演算ツリーがあることが分かる．直接依存がない演算処理はすべて並列で実行することが可能であり，並列性の高さが確認できた．これらの分析から，ソフトウェアパイプライニングによる最適化の効果が高いコードであると言える．

4.2 ベンチマーク結果と分析

MG ベンチマークを 32 レジスタモードで実行した結果をFigure 12に示す．

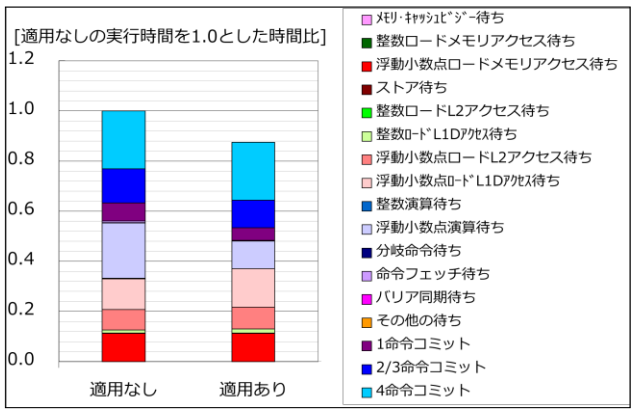

Figure 12 NPB MG の評価（全区間）

想定どおり浮動小数点演算待ちが削減されることで，実行性能が向上していることが分かる．L1 キャッシュのアクセス待ちを表す“浮動小数点ロード L1D アクセス待ち”が増加している点は，演算待ちが解消されたことで，配列データの読み込みが追い付いていないためである．また，最適化の適用後に命令数の増加を確認したが，実行性能に影響を与えるような変化はなかった．次に，高コストの処理部となっている残差計算の 2nd loop のみを測定したデータをFigure 13に示す．

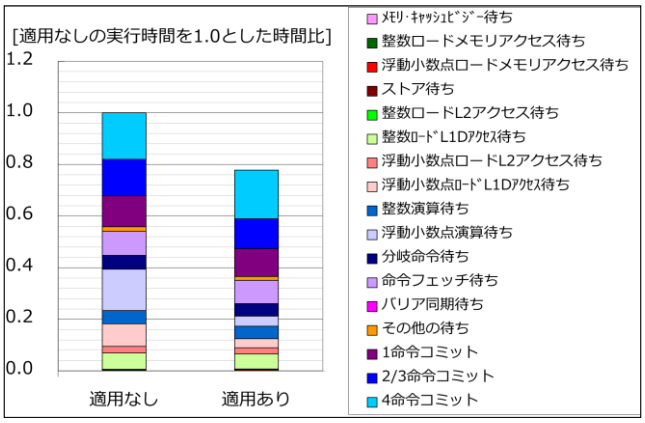

Figure 13 NPB MG の評価（ 残差計算 2nd loop）

浮動小数点演算待ちの削減効果がさらに高くなり，全体で約 22%の実行時間が削減されていることが分かる．これらのことから，MG ベンチマークの全体，特に高コストの処理区間でソフトウェアパイプライニングの効果が得られることが確認できた．

5. 数値シミュレーションによる評価

次に，数値シミュレーションをベンチマークとして，実行性能へ SIMD レジスタ数の影響評価を行う．

5.1 数値シミュレーションの概要

ポスト「京」向けに実際に開発が進められている数値シミュレーションを利用して評価を行った．ここではポスト「京」の重点課題（４）として選定されている“観測ビッグデータを活用した気象と地球環境の予測の高度化”で利用される数値気象シミュレーションコードの NICAM（Non-hydrostatic ICosahedral Atmospheric Model）[10]を利用した．

5.2.ベンチマークの概要

NICAM は，Fortran 言語で記載されたプログラムで，１時間ステップの中で大きく分けて二種類の問題を解く．一つは格子上の風速・気温・密度・水蒸気・雲物質等を有限体積法で時間発展させる力学過程である．プログラムの特徴としては，構造格子ステンシル計算であり，演算密度はあまり高くなくメモリへの要求が多くなっている．もう一つは雲の相変化や大気放射による加熱・冷却などの物理現象を扱う物理過程である．こちらはループ内の計算量が多く，分岐処理も多数含まれる．128 レジスタモードを利用し，NICAM に含まれる主要な処理区間の実行時間を測定した結果をFigure 14に示す．

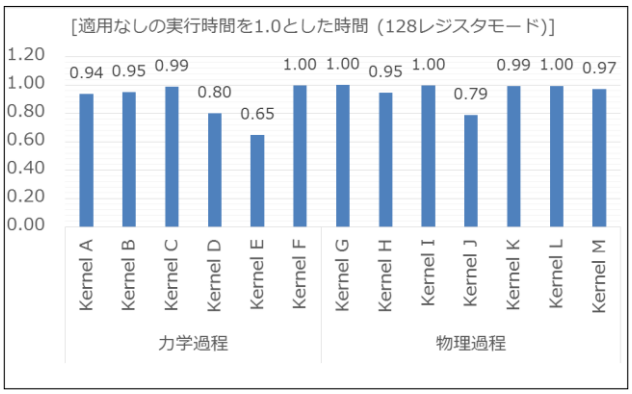

Figure 14 NICAM の評価（128 レジスタモード）

Kernel A～F は力学過程，Kernel G～M は物理過程の主要区間の実行時間を表し，ソフトウェアパイプライニングを適用しない実行時間を“1.0”とした時の相対比となっている．それぞれソフトウェアパイプライニングの適用効果が得られている．これらのデータと 32 レジスタモードで測定したデータを比較することで評価を実施した．

5.3 ベンチマーク結果と分析

NICAM の Kernel A～M に対して 32 レジスタモードで測定した結果をFigure 15に示す．

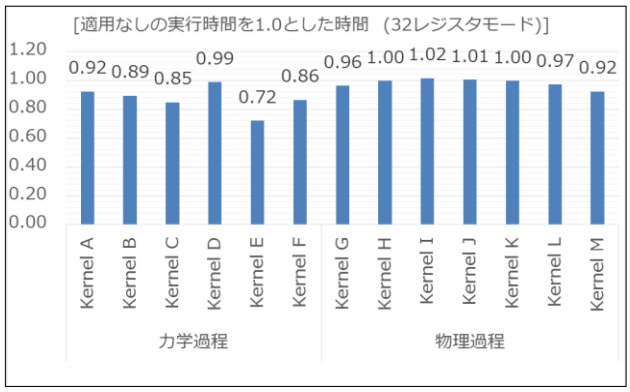

Figure 15 NICAM の評価（ 32 レジスタモード）

Figure 14と比較すると，力学過程では Kernel D と Kernel F に差があるが，それ以外は同じような効果が得られている．一方で，物理過程では Kernel J で逆効果，かつ効果に大きく差が見られる．効果のギャップが一番大きな Kernel J を分析すると，コンパイラの翻訳時のメッセージから，レジスタ不足によりソフトウェアパイプライニングが適用できていないことが分かっている．詳細プロファイラを利用し，実行時間内の浮動小数点演算待ちのコスト割合を測定すると，128 レジスタモードで 5.78%であったものが 32 レジスタモードで 22.25%に増加している．

レジスタが不足するケースにおいては，この浮動小数点演算待ちを別の手段で解消する必要があり，32 レジスタモードにおけるコンパイラに課題があると言える．

Kernel J の特徴を検証することで，この課題に対して定量的な判断材料を提案できないか検討した．処理区間ごとの処理命令の内容を比較した結果をTable 7に示す．

Table 7 処理区間ごとの実行データ

処理区間	浮動小数点演算数	有効命令数	B/F 比
Kernel A	9.22E+07	6.92E+07	1.33
Kernel B	1.53E+08	5.15E+07	1.08
Kernel C	1.05E+08	8.81E+07	1.14
Kernel D	1.00E+08	8.05E+07	1.41
Kernel E	1.88E+08	7.47E+07	0.45
Kernel F	7.70E+07	3.29E+07	1.41
Kernel G	1.63E+08	4.86E+07	1.71
Kernel H	9.49E+07	2.56E+07	4.04
Kernel I	2.90E+07	6.29E+07	1.19
Kernel J	1.51E+07	4.98E+06	1.94
Kernel K	4.74E+07	5.60E+07	8.21
Kernel L	6.10E+07	1.28E+08	1.77
Kernel M	2.03E+08	3.50E+08	3.18

実行データから Kernel J については，他に比べて有効命令数が少ないが突出した傾向は見られない．プログラムコードでは，ループボディが非常に大きく分岐処理があり，多くの除算計算を含んでいる．本研究では，Kernel J のレジスタ不足を解消し改善を検証するには至らなかったが，ポスト「京」でソフトウェアパイプライニングを活用するためには，Kernel J のコードを分析しモデル化することが課題である．

次に実行性能において一番重要な実行時間に着眼した評価を行う．そこで，128 レジスタモードと 32 レジスタモードの実行時間を比較したグラフをFigure 16に示す．グラフは，128 レジスタモードを利用した時の実行時間を“1.0”とした時の相対比となっている．

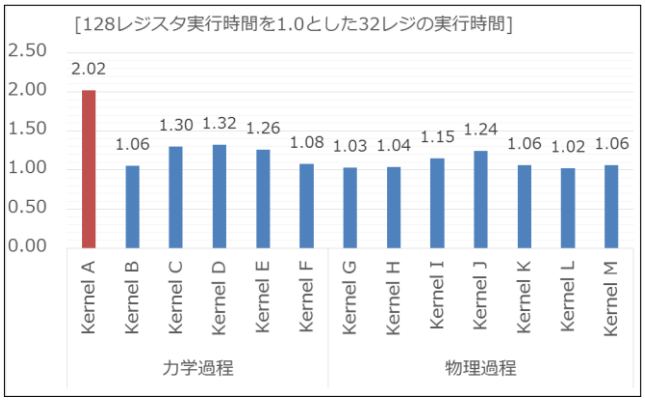

Figure 16 カーネルごとの実行時間比

全体的に 10%～30%実行時間が長くなっている．一因として，32 レジスタモードは評価用にコンパイラを作成しているため，他の最適化が効果的に動作していないことが考えられる．しかしながら，Kernel A については大きな乖離が見られる．そこで，詳細プロファイラにより Kernel A の分析を行う．Table 8に詳細プロファイラによる命令数の変化を示す．

Table 8 実行命令数の比較

	浮動小数点演算命令数	整数演算命令数	メモリロード/ストア命令数
128 レジスタモード	1.83E+07	6.34E+06	1.66E+07
32 レジスタモード	1.85E+07	7.86E+06	3.67E+07

メモリロード/ストア命令数の増加は，レジスタの不足が発生することで一時的にレジスタの内容をメモリに退避し，利用する時に復元するスピル/フィル命令が原因である．また，メモリアクセスを行うためのアドレス計算で利用する整数演算も増加している．これらは，ループボディ内で利用されるレジスタが多く，かつそれらが長い区間で利用されることでレジスタの絶対量が不足していることで発生している．高コスト部のコードの計算構成をFigure 17に示す．

```
do ij = nstart, nend
  sclt_rhogw = 配列 rhogw_vm より計算
  sclt(ij,TI) =
  (配列 rhogvx_vm, rhogvy_vm, rhogvz_vm より計算) &
  * 積算データ + sclt_rhogw
enddo
do ij = nstart, nend
  sclt_rhogw = 配列 rhogw_vm より計算
  sclt(ij,TJ) =
  (配列 rhogvx_vm, rhogvy_vm, rhogvz_vm より計算) &
  * 積算データ + sclt_rhogw
enddo
```

Figure 17 NICAM Kernel A の高コスト処理

コード内の“配列 rhogw_vm より計算”および，“配列 rhogvx_vm, rhogvy_vm, rhogvz_vm より計算”処理部の命令数が多いため，レジスタ不足が発生している．そこでループ構造を変形し，ループボディの命令数の削減を試みた．最初の変形として，ループ分割を適用してループのボディを分配した変形をFigure 18に示す．

```
do ij = nstart, nend
  sclt(ij,TI) =
  (配列 rhogvx_vm, rhogvy_vm, rhogvz_vm より計算) &
  * 積算データ
enddo
do ij = nstart, nend
  sclt_rhogw = 配列 rhogw_vm より計算
  sclt(ij,TI) = sclt(ij,TI) + sclt_rhogw
enddo
～ 以降， sclt(ij, TJ)のループも同様に分割 ～
```

Figure 18 ループ分割による変形

次にループ融合によりループボディの命令数を調整した変形をFigure 19に示す．この時，共通となる演算の削減も適用する．

```
do ij = nstart, nend
  sclt_rhogw = 配列 rhogw_vm より計算
  (共通データ X1,X2,Y1,Y2,Z1,Z2) = &
  (配列 rhogvx_vm, rhogvy_vm, rhogvz_vm より計算)
  sclt(ij,TI) = (X1,Y1,Z1)*積算データ
  sclt(ij,TJ) = (X2,Y2,Z2)*積算データ
enddo
do ij = nstart, nend
  sclt(ij,TI) = sclt(ij,TJ)+配列 rhogw_vm より計算
  sclt(ij,TI) = sclt(ij,TJ)+配列 rhogw_vm より計算
enddo
```

Figure 19 ループ融合とループ分割による変形

これらの加工により 32 レジスタモードでFigure 19のすべてのループにソフトウェアパイプライニングが適用できることを確認した．配列 sclt はループ間で利用されるため，ロード/ストア命令が増加することになるが，アドレス計算の共通式の削除やスピル/フィル命令の削減の効果の方が高く，結果としてメモリロード/ストア命令数が 3.67E+07 から 9.85E+06 まで削減された．本コードを利用し 32 レジスタモードで測定した実行時間を詳細プロファイラと重ねて表示したグラフをFigure 20に示す．

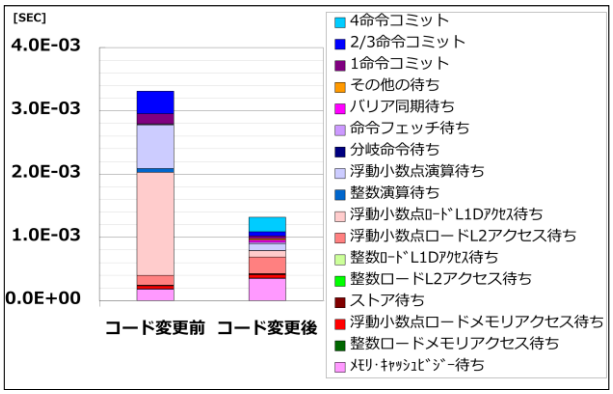

Figure 20 コード変更後の実行性能比

実行時間が大幅に短縮され，ソフトウェアパイプライニングが効果していることがわかる．浮動小数点演算待ちだけでなく，浮動小数点ロードの L1 キャッシュ待ちも大きく減少する結果となった．命令数が多くレジスタ不足によりソフトウェアパイプライニングが適用できない場合であっても，ループ変形を適切に実施し，ループ内の命令数を削減することで，省レジスタのアーキテクチャにおいてもソフトウェアパイプライニングによる性能向上が得られることが分かった．今回は，経験則からハンドチューニングによるコード変更を行ったが，ハードウェアの特性やコンパイラの最適化を考慮しハンドチューニングすることは難しい．ポスト「京」に向けてはコンパイラによる自動的な変形が必要であると考える．

NICAM の実アプリケーションを評価することで，以下の 3 つの知見が得られた．

① 実アプリケーションでも効果を観測
② 適用ケースの定量的な判断基準が必要
③ コンパイラによる自動的なループの変形が課題

6. まとめ

「京」で搭載されたコンパイラの最適化技術としてソフトウェアパイプライニングがある．本評価では，これらの技術がポスト「京」など省レジスタのアーキテクチャでも有効であることを確認した．特に浮動小数点演算待ちが発生するケースにおいて大きな効果が得られている．この効果を引き出すためにはプログラミングの段階でループ構造を考慮する必要がある．また，チューニングの段階で，プログラムのループ構造を変更することでソフトウェアパイプライニングの適用を促進できることを示した．

今後は，今回得られた知見をさらに分析し，ソフトウェアパイプライニングの適用有無に影響する要因の具体化，そして他のシステム（Arm 機，Intel 機など）における評価を行う必要があると考えている．

また，今回得た知見をポスト「京」のコデザインへフィードバックし，コンパイラ開発，およびポスト「京」向けプログラミングモデルの確立に繋げたい．

謝辞

本論文の一部は，文部科学省「特定先端大型研究施設運営費等補助金（次世代超高速電子計算機システムの開発・整備等）」で実施された内容に基づくものである．

参考文献

[1] R-CCS：ｼｽﾃﾑ紹介, http://www.r-ccs.riken.jp/jp/k/system.html
[2] Fujitsu Limited：SPARC64™ VIIIfx, http://www.fujitsu.com/downloads/JP/archive/imgjp/jhpc/sparc64viiifx-extensionsj.pdf
[3] EuroBen Benchmark：https://www.euroben.nl/
[4] R-CCS：FLAGSHIP 2020 Project, http://www.r-ccs.riken.jp/fs2020p/
[5] arm Developer：Introduction to Scalable Vector Extention, https://developer.arm.com/products/software-development-tools/hpc/sve
[6] MACOM：X-Gene™ Family, https://www.apm.com/products/data-center/X-Gene-family/
[7] Rau,B. R. and Glaeser, C. D., "Some Scheduling Techniques and An Easily Schedulable Horizontal Architecture for High Performance Scientific Computing",Proceedings of the 14 th Annual Workshop on Microprogramming, pp.183-198, 1981
[8] Lam, Monica, "Software pipelining: an effective scheduling technique for VLIW machines", Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation, pp. 318-328, 1988
[9] Llosa, Josep, "Swing Modulo Scheduling: A Lifetime-Sensitive Approach", Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, pp.80-87, 1996
[10] NPB：https://www.nas.nasa.gov/publications/npb.html
[11] NICAM：http://nicam.jp/hiki/