実行前試行によりローカルメモリを持つマルチプロセッサを有効利用する環境: EULASH

山本 淳二, 服部 大[1], 鬼頭 宏幸, 山口 喜弘, 天野 英晴

慶應義塾大学理工学部

本論文では高速なローカルメモリとアクセスに大きなレイテンシを伴う共有メモリの双方を持つマルチプロセッサを最適に使用するための環境 EULASH を提案する。EULASH はプリプロセッサシステム、カーネル、変数の管理と同期のサポートをするライブラリからなる。

EULASH においてソースプログラムは共有変数のみを用いる軽量なスレッドを用いて記述される。このプログラムをプリプロセッサシステムがユーザの助け無しに両メモリシステムを効率良く使用するように再構成する。実行前試行を用いた最適化により 50% から 90% の共有変数が共有メモリを必要としない型に変換される。また、評価の結果、最適化を行うことでスイッチ結合型マルチプロセッサ上で実行時間が 13% から 20% 短縮された。

EULASH: an environment for efficient use of multiprocessor with trial run

J. Yamamoto, D. Hattori[2], H. Kitoh, Y. Yamaguti, H.Amano

Faculty of Science and Technology, Keio University

The EULASH environment is proposed for making the best use of a multiprocessor with a high speed local memory and shared memory with a large latency. It consists of the preprocessing system, the kernel, and the library for management of variables and synchronization.

On the EULASH, a program is written with light weight threads using only shared variables. The preprocessing system restructures the program in order to use both memory system efficiently without user's optimization. By the optimization using the *trial run*, between 50% and 90% of shared variables are converted so as not to use the shared memory. From the result of evaluations the execution time with optimization is about 13%–20% better than that without optimization on a switch connected multiprocessor.

---

[1] 現在 松下電器産業 (株) に勤務

[2] Now he had joined to Matsushita electric industrial CO.,LTD.

1 まえがき

ソフトウェア環境 EULASH (Environment for Using Local And SHared memory) は次の 2 つのメモリシステムを持つマシンをターゲットとする。ひとつは高速にアクセスのできるローカルメモリ、もう一つは大きなレイテンシを伴うが全てのプロセッサからアクセスできるリモートメモリである。このタイプのマシンには、複雑なディレクトリキャッシュを持たない単純な NUMA 型システムや、NYU Ultracomputer[1] や BBN Butterfly TC2000[2] のようなスイッチ結合型マルチプロセッサが含まれる。これらの計算機上で高速に実行を行おうとするならば、高速なローカルメモリを効率的に使用することが不可欠である。もしプログラマがプロセスと変数の割当を注意深く行うならば共有メモリの使用を最小限にとどめることができる。しかしながら、このことはプログラマに取って大きな負担となる。 EULASH は容易なプログラミングとこれらのマシン上で単一ジョブの高速実行を提供するものである。

EULASH では、プログラムは共有変数だけを使用する軽量なスレッドとして記述される。コンパイラはスレッドと変数の関係を解析し、依存関係のグラフを作成する。その後、小規模なデータによる「試行」により、各変数、各スレッドのアクセス情報を計測する。これらの情報を元に、スレッドを仮想プロセッサに割り当て、スレッドのコンテキストスイッチと共有メモリアクセスによるオーバヘッドが最小になるようにプログラムを再構成する。また、プログラム中の変数を分類し、ローカルメモリや共有メモリに配置する。実行時には、高速なスレッドのコンテキストスイッチングと仮想記憶システムを提供するカーネルにより、プログラムを実行する。

これらの機構により、ユーザによる最適化なしに並列アプリケーションプログラムがローカルメモリ、共有メモリの双方を効率的に使用することができる。

本稿では、EULASH のコンパイラ、カーネル、実行時ライブラリの提案と実マシン上での評価を行う。

2 EULASH の概要

容易なプログラミングと単一ジョブの高速実行をサポートするために、EULASH では単純であるが強力なプログラミングモデルと実行モデルを提供する。

EULASH の環境では、プログラマはアプリケーションプログラムを Mach や UNIX 上の C-threads ライブラリ等のマルチスレッドライブラリと似た記法で記述する。この時、プログラマはデータの配置やターゲットマシンの構造について考慮する必要は無い。しかし、EULASH では、もしプログラマ自身がデータの配置や、スレッドのグループ化、コンテキストスイッチの方法について特に規定したい場合には、そのような記述を行うことができる。

コンテキストスイッチによる損失と大きなレイテンシを伴う共有メモリへの参照を最小限にするために、EULASH の前処理系では、静的な解析と「試行」を用いる。

スレッドと変数の関係は静的に解析される。プログラマによって記述されたスレッドはプロセスにまとめられる。 EULASH ではプロセスとは仮想プロセッサであり、これは静的にマッピングされる。大抵は実プロセッサに 1 対 1 に割り当てられるが、パーティションの変更やシステムのエラーによってターゲットマシンのプロセッサ数が変更になった場合などは 1 実プロセッサに複数のプロセスが割り当てられることがある。

結果として、単純なプログラミングモデル (図 1(a)) は図 1(b) のように再構成される。

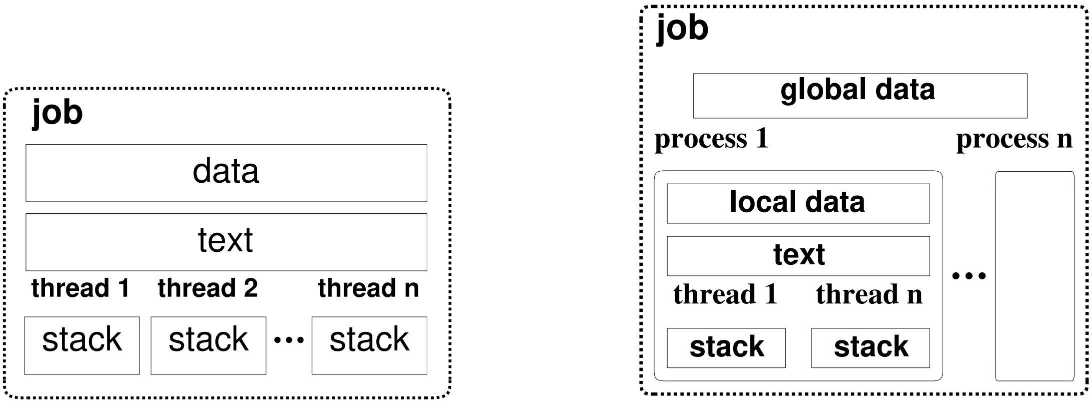
変数参照のレイテンシを隠蔽するために、ローカル型、共有型、書き込み無効化型、書き込み更新型、モービル型の 5 つの変数型をサポートする。しかし、各変数がどのようにプロセスから参照されるのかを静的に解析することは難しいため、「試行」を行う。「試行」ではプログラムの核となる部分を実行し、変数の参照情報を収集する。この情報を基にコンパイラは変数の型を決定する。

再構成され最適化されたプログラムは、高速なスレッド・プロセス管理と仮想記憶管理を行う EULASH カーネル上で実行される。

EULASH システムの流れは図 2のようになっている。

3 前処理系

EULASH の前処理系での主な処理は次の通りである。



(a) model for user　　(b) restructured model

図 1: プログラミングモデル

- プログラマによって記述された単純なマルチスレッドモデルのプログラムを実行時にスレッドの管理によるオーバヘッドが小さい構造へと変化する。
- 共有メモリアクセスによるレイテンシの影響を最小限にするために高速なローカルメモリを使用する。

3.1 プログラムの再構成

EULASH では、block 法、cyclic 法、データ依存法、緊密スレッド優先法、の 4 つのスレッドのグループ化の方法を提供する。

block 法

スレッドは生成順にグループ化され、プロセスに割り当てられる。(図 3.1(a))

cyclic 法

スレッドは生成順に 1 つ 1 つプロセスに割り当てられる。(図 3.1(b))

データ依存法 (DD)

まず、スレッド t が同じ変数を共有する場合に枝 s で接続されるような依存グラフ D を作成する。最初に生成されるスレッドをマスタスレッドとする。次にマスタスレッドからもっとも遠くに位置するスレッドであるサブマスタスレッドを m 個選ぶ (プロセスの数は $m+1$ である)。マスタスレッドおよびサブマスタスレッドはそれぞれ別のプロセスに割り当てられる。次に、マスタ / サブマスタスレッドに隣接するスレッドをマスタ / サブマスタスレッドと同じプロセスに割り当てる。全スレッドの割当が決定するまで、同様にして割り当てるスレッドを決定する。

緊密スレッド優先法 (TCTF)

この方法では各変数について参照するスレッド数をカウントする。まず、もっとも参照するスレッド数が大きい変数を選び出し、その変数を参照するスレッドを同じプロセスに割り当てる。その後、全てのスレッドが割り当てられるまでこの作業を繰り返す。

3.2 変数型

スレッドのグループ化の後、前処理系はそれぞれの変数の型を決定する。変数は 2 つの通常型と 3 つの特殊型、計 5 種類の型を持つ。通常型はローカル型と共有型で、これらは静的にローカルメモリ、共有メモリに配置される。

特殊型は書き込み更新型、書き込み無効化型、モービル型である。書き込み更新型は、初め共有メモリに配置され、アクセスがあった時にローカルメモリ

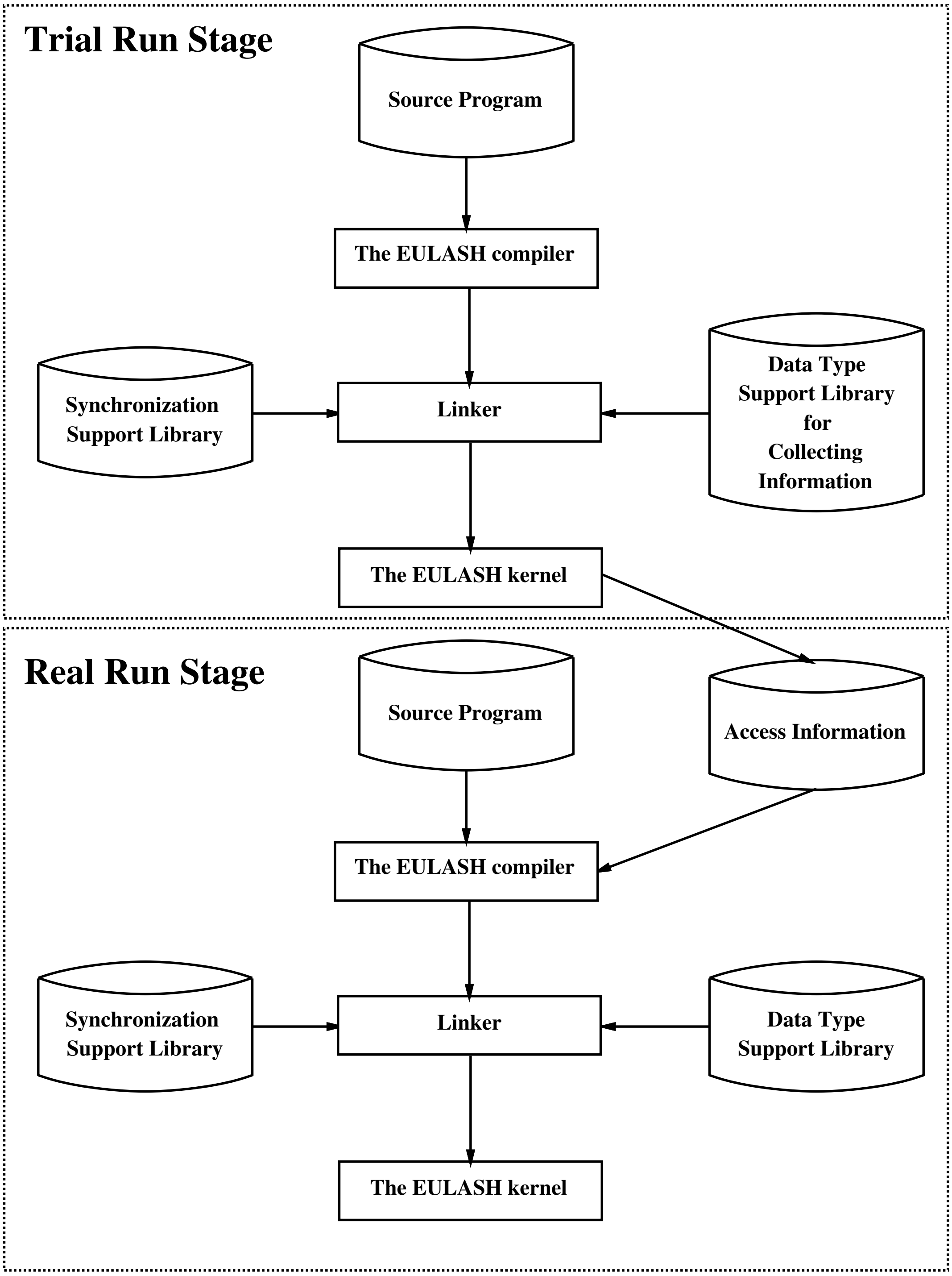
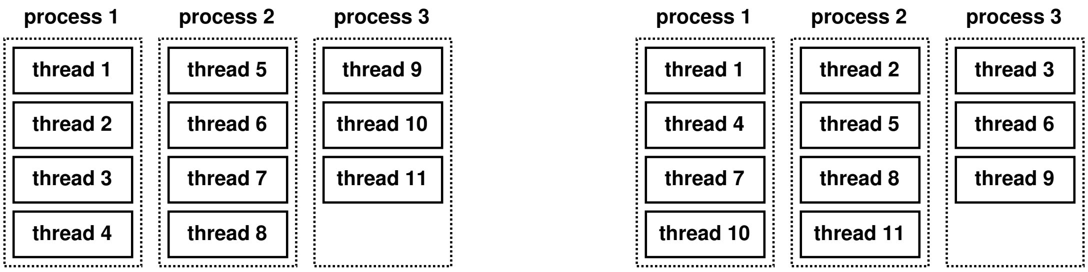


図 2: EULASH での実行手順



(a) Block assignment (b) Cyclic assignment

図 3: スレッドの割り当て法

にコピーされる。この変数の更新時には情報が放送され、コピーされた全ての変数が更新される。書き込み無効化型も初めは共有メモリに配置され、アクセスがあった時にローカルメモリにコピーされる。しかし、書き込み更新型と違い変数の更新時には他のコピーを無効にする。書き込み更新型 / 無効化型を使用することで、ローカルメモリはキャッシュラインサイズが変数のサイズであるようなソフトウェアキャッシュとして使用することができる。

モービル型は初めローカルメモリに配置される。他の特殊型と違い、アクセス時にコピーされることは無い。他のプロセッサからアクセスされた時には共有メモリに移動する。初めに配置されたプロセスからのみアクセスがされる間は共有メモリにアクセスする必要が無い。

変数の型を決定するためには変数へのアクセス情報が必要である。この情報は後述する「試行」ステージによって収集される。

型の決定は次の順序で決定される。

1. ひとつのプロセスからしかアクセスされない変数は ローカル型とする。
2. リードアクセスだけが行われる変数は ローカル型とする。フォークが行われる前に、初期化のためにアクセスされても良い。
3. 特定のプロセスからのアクセス頻度が閾値 `Th1` を越えるならば、書き込み無効化型とする。
4. 全アクセスに比べリードアクセスの割合が閾値 `Th2` を越えるならば、書き込み更新型とする。
5. スレッドが終了するまでにそのスレッドからだけアクセスされる変数はモービル型とする。前のスレッドの終了後は別のスレッドがその変数にアクセスしても良い。
6. 上記以外の変数は 共有型である。

3.3 試行

コンパイラによる最適化のために、変数参照の情報を収集する「試行」が行われる。「試行」の時間を削減するためにイテレーション回数やデータのサイズを縮小して行われる。「試行」ではプログラムは次の条件で実行される。

- 全ての変数は共有型とする。
- スレッドのプロセスへの割り当ては負荷のバランスのみ考慮される。
- 変数へのアクセス時にライブラリが呼ばれ、アクセスの回数と種類が記録される。

「試行」後、収集した情報を基にコンパイラはスレッドの割当と変数の型を決定する。情報の収集を行う場合の実行時間は収集をしない場合に比べ、1.5 倍ほどである。

4 実行時システム

EULASH カーネルはローカルメモリを持つマルチプロセッサ上でシングルジョブを実行する軽量な OS である。ユーザモード・スーパバイザモード間の遷移に伴うオーバヘッドを減らすためにできるだ

けユーザモードで実行されるライブラリで実装する。このことにより、マシン依存部分はカーネル本体だけになり移植性が向上する。

たとえば、変数にはローカル型、共有型、書き込み更新型、書き込み無効化型の5つがある。ローカル型と共有型は静的にローカルメモリと共有メモリに配置されるが、その他3つの型の変数にアクセスをするには実行時にサポートルーチンが必要である。モード遷移によるオーバヘッドを減らすためにこれらのサポートルーチンはカーネル内ではなくライブラリとして提供をする。そのため、EULASH カーネルの行う主な処理はメモリ管理とジョブ管理である。

メモリ管理　一般的にローカルメモリの総量は大きくても、ノードひとつあたりのローカルメモリの容量は限られている。そのため、EULASH ではローカルメモリのスワップエリアとして共有メモリを使用する。もし、共有メモリがフルならば、さらに2次記憶にスワップアウトする。

ジョブ管理　前述したように、ユーザが記述した簡単なスレッドベースのモデルによるプログラムをコンパイラが高速に実行できるプロセス·スレッドの構造に変換する。プログラムはシングルプロセス、シングルジョブで実行が開始される。“Process fork” が実行されるとプロセスが生成され、各プロセッサに静的に割り当てられる。“Thread fork” ではスレッドが同じプロセッサ上に生成される。プロセス·スレッドは一度生成されるとマイグレートすることはない。同じプロセス内のスレッドは全メモリ空間を共有する実行主体である。スレッドのコンテキストスイッチはロック待ちなどそのスレッドが実行できなくなった場合や明示的に指示された場合に行われる。また、同期操作のない無限ループによるデッドロックを回避する為にタイムアウトによるコンテキストスイッチも行う事が出来る。プログラマによる独自のスケジューラをカーネルに組み込む事も容易であるが、デフォルトではラウンドロビンによるスケジューリングを行う。

5　評価

5.1　ターゲットマシン

EULASH のターゲットとしているマシンは、コヒーレントキャッシュを持たない大規模 NUMA やスイッチ結合による UMA である。現在、EULASH はバス結合型並列計算機テストベッド ATTEMPT-0[3] と、スイッチ結合型マルチプロセッサ SNAIL[4] で稼働している。

ATTEMPT-0(図4) では10プロセッサと共有メモリシステムが IEEE Futurebus によって結合されている。それぞれのプロセッサボードには CPU/FPU (MC68030/MC68882)、ローカルメモリ、ライトスルーのスヌープキャッシュが搭載されている。また、シンクロナイザと呼ばれる同期·交信機構を持つ。シンクロナイザでは Fetch & Decrement と同期動作に伴う割り込み機構を備えている。

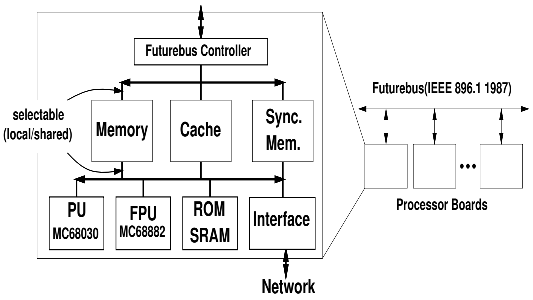


図 4: ATTEMPT-0 のブロック図

表1,2はそれぞれメモリシステムとシンクロナイザのアクセス時間を示している。表1から ATTEMPT-0 ではスヌープキャッシュの働きにより共有メモリアクセスの時間がローカルメモリより速いか同じ程度である事がわかる。このことから ATTEMPT-0 は本来 EULASH のターゲットマシンではないが、最適化を行う上で必要な様々な測定が出来る為 EULASH は ATTEMPT-0 上で開発された。

一方の SNAIL は EULASH のターゲットの一つであるスイッチ結合型のマルチプロセッサである。SNAIL (図5) では16プロセッサと16共有メモリモジュールが SSS-MIN(Simple Serial Synchronized-Multistage Interconnection Network) と呼ばれる高速な MIN によって結合されている。プロセッサボー

表 1: メモリへのアクセス時間 (ATTEMPT-0)

メモリシステム	種類	時間 (ns)	転送サイズ
local memory		250	4 bytes
shared memory	Read miss	182000	64 bytes
	Read hit	150	4 bytes
	Write	750	4 bytes

表 2: シンクロナイザへのアクセス時間 (ATTEMPT-0)

機能	時間 (ns)	バスオペレーション
Read	200	no
Write	650	yes
Fetch&Dec	750	yes

ドそれぞれに 4 つの CPU (MC68040) とローカルメモリが搭載されている。共有メモリシステムでは Fetch & Decrement と Test & Set のような同期機構がサポートされている。表 3ではメモリシステムに対するアクセス時間を示している。この表からわかるように SNAIL では共有メモリのアクセス時間がローカルメモリに対するそれより 5 倍程大きい。また、MIN が混雑をすると共有メモリのアクセス時間はこの表の値より大きくなり得る。

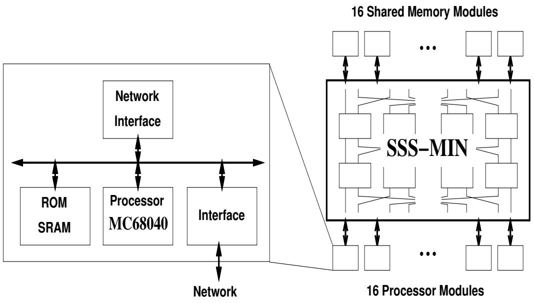


図 5: SNAIL のブロック図

表 3: メモリのアクセス時間 (SNAIL)

memory system	access type	time(ns)
local memory	read/write	250
shared memory	read	1375
	write	250

5.2 最適化の効果

プリプロセッサシステムによる最適化の効果の評価は次の 2 つのプログラムによって行った。

- **Jacobi:** ヤコビ法による連立 1 次方程式の求解。
- **Heat:** 差分法による熱伝導方程式の求解。

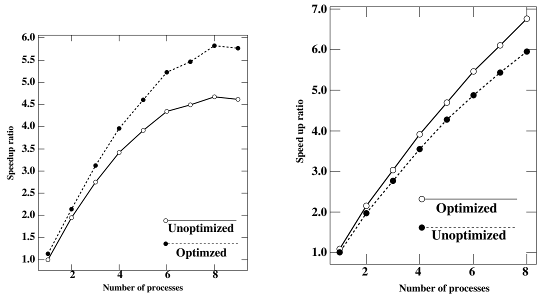
変数の型の選択に用いる閾値は以下の様にした。

- **Th1:** スレッドのアクセスが全スレッドによるアクセスの 90% 以上の場合、書き込み無効化型とする。
- **Th2:** リードアクセスが全アクセスの 70% 以上の場合、書き込み更新型とする。

Jacobi では、グループ化の方法によらず全データの 60% が「ローカル型」となり、残りが「書き込み更新型」となった。「ローカル型」として選択されたデータは係数行列であり、これは初期化時に唯一のスレッドから書き込まれるだけのデータである。そのほかの変数は書き込みに比べ読み出しが頻繁に行われるため、プリプロセッサシステムによって自動的に「書き込み更新型」が選択される。

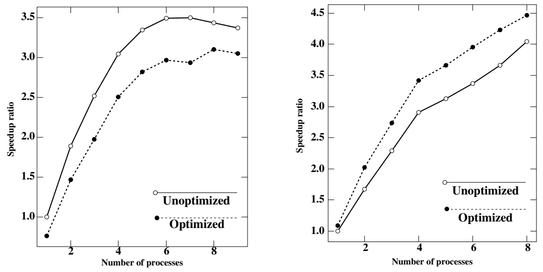
図 6は ATTEMPT-0 と SNAIL において **Jacobi** の最適化の効果を示している。スヌープキャッシュによって共有メモリに対するレイテンシが小さくなる ATTEMPT-0 においてでさえ最適化を行わない場合に比べ 30% ほど速くなる。SNAIL では最適化を行った場合 13% 高速になる。図 7に **Heat** の最適化の効果を示す。ATTEMPT-0 ではかえって速度が低下していることがわかるが、これはリードアクセスにおいてスヌープキャッシュにヒットした場合は共有メモリの方がローカルメモリより高速な為である。**Heat** ではキャッシュのヒット率が非常に高く、このようなプログラムでは変数の管理機構を省くことの出来ない書き込み更新型より共有メモリアクセスの方が高速になる。しかし、我々がターゲットマシンとしている SNAIL では最適化の結果 22% 高速になることがわかる。

図 8に ATTEMPT-0 における各スレッド割り当て法によるスピードアップ率を示す。この図から、Block と TCTF 法が Cyclic, DD 法より優れていることがわかる。これは、Block, TCTF 法を用いることで他の方法に比べ「ローカル」変数が 10 から 20% ほど増加することによる。(Cyclic, DD 法では 10-20% がローカル変数となる。)



(a) ATTEMPT-0　(b) SNAIL

図 6: スピードアップ率 (Jacobi)



(a) ATTEMPT-0　(b) SNAIL

図 7: スピードアップ率 (Heat)

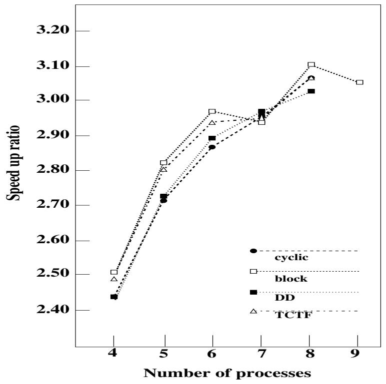


図 8: スレッド割り当て法の比較 (Heat / ATTEMPT-0)

6 結論

高速なローカルメモリと低速な共有メモリを持つ並列計算機を効率良く使用するソフトウェア環境 EULASH を提案した。EULASH はコンパイラ、カーネルと変数の管理、同期のサポートを行うライブラリからなる。

試行」を行うことでコンパイラは 10% から 20% の変数をローカルメモリに配置し、他の変数も書き込み無効化型や書き込み更新型に分類する。この最適化により、スイッチ結合型並列計算機において 13% から 20% のパフォーマンスの向上を得ることが出来た。

参考文献

[1] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. Mcauliffe, L. Rudolf, and M. Snir, “The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer,” IEEE Transaction on Computer Systems, Vol. c-32, No.2, 1983.

[2] BBN Laboratories, “Butterfly (TM) Parallel Processor Overview,” BBN Computer Company, Cambridge, MA, 1st edition, June 1985.

[3] H. Amano, T. Terasawa, and T. Kudoh, “Cache with synchronization mechanism,” Proceedings of IFIP Congress89, pp.1001–1006, Aug. 1989.

[4] M. Sasahara et al. “SNAIL: A Multiprocessor Based on the Simple Serial Synchronized Multistage Interconnection Network Architecture,” Proceedings of the 1994 International Conference on Parallel Processing, Vol.I, pp.I-117–I-120, Aug. 1994.

[5] H. Amano, L. Zhou, and K. Gaye, “SSS(Simple Serial Synchronized)-MIN: a novel multi stage interconnection architecture for multiprocessors,” Proceedings of the IFIP 12th World Computer Congress, Vol. I, pp.571–577, Sep. 1992.