ハイパフォーマンスコンピューティング 60－2
（1996．3．4）

超並列計算機SR2001による最短経路問題の解法について

江丸 裕教† 高井 昌彰††
†北海道大学工学部，††北海道大学大型計算機センター

科学技術の急速な進歩により、現在、最高性能を持つ計算機がベクトル型のスーパコンピュータから超並列計算機へと移行しつつある。しかしながら、ベクトル型のスーパコンピュータに得意な問題と不得意な問題があるように、全ての問題に対して超並列計算機が高い処理能力を発揮できるわけではない。本論文では並列計算向きの非数値的応用問題の1つとして最短経路問題を取り上げ、この実装を通して超並列計算機SR2001の基本性能について考察を行なった。Floydのアルゴリズムを ParallelWare の CUBIX モデルで並列化した場合、ノード間通信の実装方法を工夫することによって、理想的なスピードアップ率が得られ、優れたスケーラビリティを示すことが確認された。

Performance Evaluation of the SR2001 Massively Parallel Processor by the Shortest Path Problems

Hironori Emaru†, and Yoshiaki Takai††,
†Faculty of Engineering, Hokkaido University
††Computing Center, Hokkaido University

Computer architectures for high-performance computing have been shifting from conventional vector supercomputers toward massively parallel processors (MPPs). However, it is true that the MPPs are not good for all kinds of applications at hand. In this paper, we evaluate basic computing performance of Hitachi's SR2001 by using a large shortest path problem as a parallel processing-oriented nonnumerical application. The SR2001 is a kind of MPPs based on a distributed memory architecture. A parallelized Floyd's algorithm implemented in a CUBIX model of ParallelWare (EXPRESS) achieves almost ideal speedup and good scalability.

1 はじめに

近年のコンピュータの計算能力の急速な進歩により、最高性能を持つ計算機がベクトル型のスーパコンピュータから超並列計算機へと移行しつつある。しかしながら、ベクトル型のスーパコンピュータに得意な問題と不得意な問題があるように、全ての問題に対して、超並列計算機が高い処理能力を発揮できるわけではない。

また、すでに基本的なアーキテクチャが確立しているベクトル型スーパコンピュータとは異なり、超並列計算機に関しては、現在もさまざまなアーキテクチャが提案、研究されている段階である [1]。

本論文では、最短経路問題の解法を例題として取り上げ、超並列計算機 SR2001 の特性について考察する。

まず最初に、SR2001 のハードウェア、ソフトウェアを含む開発環境について、簡単に解説する。次に今回用いた、最短経路問題の解法のアルゴリズムと、その並列実装について述べる。最後に、実装と評価についてさまざまな角度から考察を行なう。

2 超並列計算機 SR2001 とは

今回評価実験に用いた、SR2001 のハードウェア仕様を図 1 に示す。

本機においては開発環境として ParallelWare (EXPRESS) が提供されている。これはノード間の通信ライブラリおよび、並列化支援ツールからなる [2, 3]。

- メッセージパッシング型プログラミングのための通信ライブラリ
- パフォーマンスモニタ、シンボリックデバッガなどの並列化支援ツール

これにより、C あるいは FORTRAN プログラムに並列処理用の ParallelWare 関数を組み合わせることによってメッセージパッシングを用いた並列処理プログラムを作成することができる。ただし、自動並列化の機能は提供されていない。

ParallelWare においては、ノード間通信は基本的に 1 対 1 で行なわれるが、ライブラリにより PE 数を N とした時に $O(logN)$ で高速にブロードキャストを行なうこともできる。

プログラミング環境としては、HOST-NODE と CUBIX という二つのモデルが用意されている。CUBIX モデルでは、基本的に全てのノードが同じプログラムを実行する。一方、HOST-NODE モデルでは、HOST ノードが他の各ノードを管理することによって、ノードごとに異なるプログラムを実行することができる。

HOST-NODE の特徴として、各 PE の同期が必要なく、MIMD としての性能が発揮しやすいこと、入出力の制御が容易なことなどが挙げられる。

CUBIX の特徴としては、各 PE で同期をとる必要のあるプログラムが書きやすく、SPMD(Single Program Multiple Data) 型の大規模な計算に向いていることが挙げられる。

各 PE が同じようなデータを扱い、同じような計算をする場合は CUBIX モデルが向いていると考えられるので、本稿では CUBIX モデルによって最短経路問題を解くことを中心に考える。

3 最短経路問題

3.1 最短経路問題とは

最短経路問題は、グラフ問題の一つであり、重みつきグラフ $G = (V, E)$ が与えられた時に、任意の 2 頂点を結ぶ経路の中から、辺の重みの総和が最小のものを求める問題である。

最短経路問題は、解を求める範囲によって三つのクラスに分けることができる。

1. 出発点 A から目的地 B への最短経路を求める
2. 出発点 A から V の中の各頂点への最短経路のコストを全て求める
3. V の中の全ての頂点対の最短経路のコストを求める。

アーキテクチャ	分散メモリ型 MIMD
CPU	PA-RISC 90MHz
ノード数	64（最大 128）
理論ピーク性能	11.5GFlops(180MFlops*64)
主記憶容量	8.2GByte(128Mbyte*64)
ノード間結合網	二次元クロスバー (通信速度 100Mbyte/sec)
通信方式	send/receive 型のメッセージパッシング

図 1: SR2001 のハードウェア仕様

2 番目のクラスの問題に対するアルゴリズムとして代表的なものがダイクストラのアルゴリズムであり、3 番目のクラスのアルゴリズムとしてはフロイドのアルゴリズムがある [4]。

今回、対象とする問題は以下の条件を満たすものとする。

- 全ての頂点対について最短経路のコストを求めること。つまり、上記の分類では 3 番目のクラスに相当する。
- グラフが、密結合であること。つまり、頂点の数を N とした時に、辺の数が $O(N^2)$ であること。
- 有向グラフであること。

以上の条件より、2 頂点間の辺の重み（経路情報）は頂点数を N とした時に $N \times N$ の 2 次元配列で与える。

3.2 並列化手法

本節では、二つのアルゴリズム (Floyd と Dijkstra) を SR2001 に実装する手法を述べる。

3.2.1 Floyd の並列化

最初にフロイドのアルゴリズムの C によるプロトタイプを示す。配列 v は経路情報を持つ 2 次元配列として与えられ、最終的に最短経路のコストを返す。

```
for(k=0;k<N;k++){
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
    if(v[i][k]+v[k][j] < v[i][j])
        v[i][j] = v[i][k]+v[k][j];
        }
    }
}
```

次に以下の 2 点を考慮して、データ分割について考える。

- データ分割を PE 間でどう行なうか。
- 重みの比較処理に際し、データの正当性をどのように保証するか。

このアルゴリズムでは、内側の二つのループ i、j は任意の順序で実行することができるが、一番外側の k のループに関しては、逐次的に行なうことを保証しなければならない。

したがって、1 行、または 1 列を単位として、各 PE に、頂点間の重みを表した配列を分配し、図 2 のように計算を行なうことにする。

列の比較対象要素（v[i][k] にあたる）は、各 PE がもっている分で足りるため、k 行目を全 PE にブロードキャストするだけで良い。したがって、N 頂点の最短経路問題を P 個の PE で解くときアルゴリズムの骨格は次のようになる。

```
各 PE に N/P 行の重みデータを割り当てる
for(k=0;k<N;k++){
    担当 PE が k 行目をブロードキャスト
    各 PE が自分の担当する部分について
```

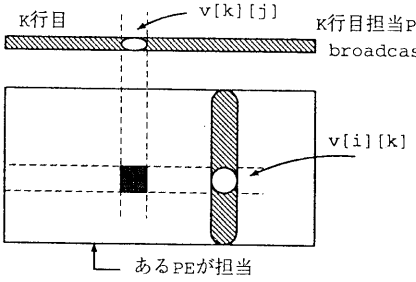

図 2: 各 PE の計算

以下の計算を行なう。

```
        if(v[i][k]+v[k][j] < v[i][j])
          v[i][j] = v[i][k]+v[k][j];
}
```

3.3 Dijkstra の並列化

ダイクストラのアルゴリズムを並列化するために、ここでは各 PE に出発点を割り当て、各 PE が割り当てられた出発点に対する最短経路問題を解くことにする。こうすることにより PE 間の通信の必要は全くなくなるが、データの分散がはかれないため、各 PE で同じデータを重複して持たなければならなくなる。

4 実装と考察

前章で取り上げた二つの最短経路問題のアルゴリズムを SR2001 上で実装・評価するにあたり、以下の点から考察を行なう。

- 並列化に関して
  - 効率の良いノード間通信
  - タスクのロードバランス
  - 並列化したことによるオーバヘッド
- 一般的なプログラミング技法に関して
  - RISC 向きのアルゴリズムの選択
  - オブジェクトの最適化

4.1 通信

4.1.1 プログラムにおける実際の通信コスト

SR2001 の基本的な通信方法は最初に述べたように send/receive 型のメッセージパッシングである。今回、最短経路問題をフロイドのアルゴリズムを用いて解く上で用いる通信方式は、通信元の PE をサイクリックに変えながら、ブロードキャストを頂点の個数分繰り返すというものである。

このような場合には、通信ライブラリに用意されているブロードキャスト関数を用いるよりも、通常の exread/exwrite 関数（ParallelWare が提供する 1 対 1 の同期通信）を用いた方が通信のパフォーマンスを向上させることができる。

これは、ブロードキャスト関数を用いた場合、基本的には全 PE が封鎖されてしまうのに対し、各 PE に対して個別に send/receive を行なうようにすると、全体的な封鎖が起こらないためである。

図 3 に通信時間の比較を示す。これはフロイドのプログラムの通信の部分のみを切り出したもので、それぞれ PE 数が 2 の場合の通信時間で正規化している。

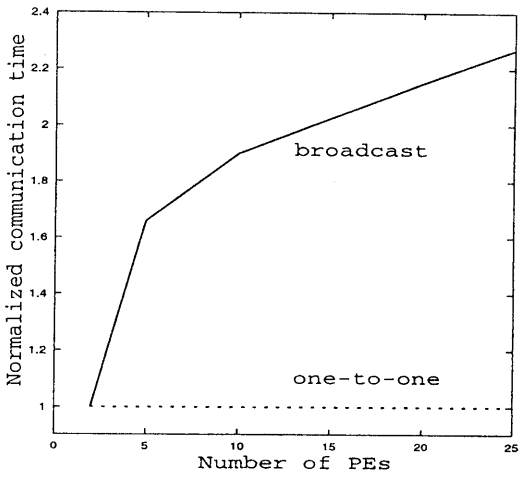

図 3: 通信時間の比較

ブロードキャスト関数を用いると、PE 数を N としたとき $log_2 N$ のオーダで通信時間が増加するが、exread/exwrite を用いることによっ

て、PE 数の増加に対して通信を定数時間に抑えることができる。

4.1.2 通信性能について

SR2001 とその上に実装された ParallelWare のもつ通信性能を明らかにするために、いくつかの実験を行なったので、その結果を述べる。

まず、通信の立ち上がり（オーバヘッド）であるが、一回の通信につきおおよそ 180μ sec である。これは、オプションを設定することによっておおよそ 66μ sec にまで減らすことが可能である。

またスループットに関しては、約 2Kbyte 以上のデータを一括して送る場合に、最大の効率が得られ、約 17Mbyte/sec となる。ParallelWare の実装が、通信時に一度、バッファにコピーしてから送受信を行なうようになっているため、ハードウェア仕様の約 1/6 の性能しかでていない。

4.2 オブジェクトの最適化

SR2001 の C コンパイラは cc のレベルでのみ最適化を行ない、並列化に関する部分や、通信に関する部分に関しては最適化は行なわない。そのため、最適化オプションの効率を見るために、srcc（ParallelWare を用いた時の並列用 C コンパイラで最終的には cc をコール）ではなく、直接シングルプロセッサの cc において実験を行なった。

以下の機種と各コンパイラの組み合わせで、1000 頂点の最短経路問題を Floyd のアルゴリズムによって解く時間を測定する。

機種	CPU	クロック
JP4	PowerPC604	100MHz
SS20	Super SPARC	66MHz
indy	R4600	100MHz

図 4 は比較の結果である。最適化オプションは、各コンパイラで最も良さそうなものを選んだ。また図にも示した通り測定時間の単位は秒であり、測定した時間は入出力などを除いた、純粋な計算時間のみである。これは以降の比較においても同じである。

この結果から、SR2001 に使われている CPU(PA-RISC) は、通常のワークステーションのものと同程度の性能を持っていることがわかる。しかしながら、最適化の度合は、今回実験したものの中では最悪であった。

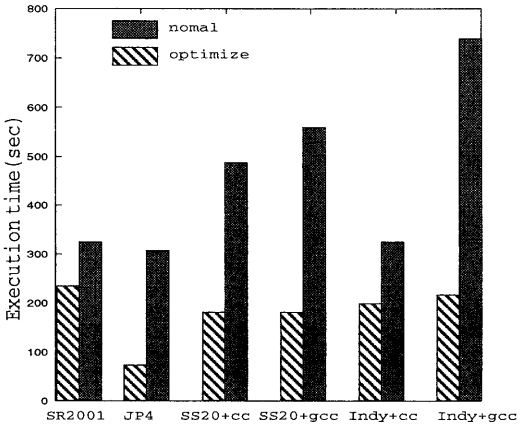

図 4: シングルプロセッサにおける最適化の効果

4.3 並列処理による高速化

1000 頂点の最短経路問題を SR2001 で並列処理した場合の計算時間を図 5 に示す。

この実験では、各 PE における通信時間の割合が計算時間に比較して十分小さいため（10PE で約 3%)、シングルプロセッサとほぼ同じ最適化効率が得られた。また、フロイドの実装においては、通信時間が PE 数によらず、ほぼ一定となるような通信方式を用いたため、理想に近いスピードアップ率が得られた。

ダイクストラのアルゴリズムによる実装では、PE 間の通信が全く無いにも関わらず、全体的に時間がかかっている。計算のオーダでは頂点数を N とした時に $O(N^3)$ となるので両者は同じであるが、定数項の部分の差があるためと思われる。

4.4 問題サイズに関して

本稿で述べた実装方法で、規模の大きい問題を考えた場合、ダイクストラでは解けないのだが、フロイドでは解けるというサイズの問題が

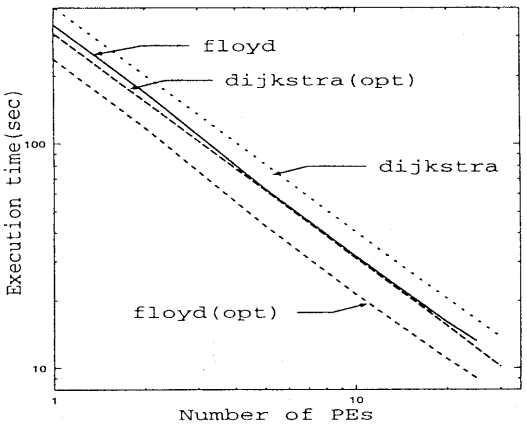

図 5: 計算時間の比較

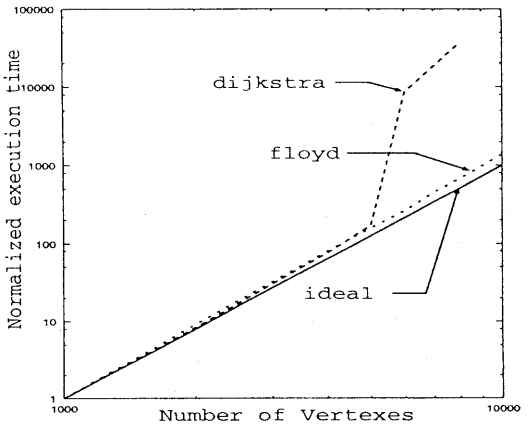

図 6: 異なる問題サイズに対する二つのプログラムの実行時間 (PE 数 10 台)

ある。これは二つの実装が以下のように異なっているためである。

Dijkstra : 各 PE が配列データの全てを持つ。
Floyd : 配列データの $\frac{1}{PE}$ だけを持つ。

本研究で用いた SR2001 は各 PE 毎に 128 Mbyte のメモリを持っているため、整数型の 2 次元配列の場合、各 PE において、5000 頂点まではメモリ上に持つことが可能である。しかし 6000 頂点になると、$6000^2 \times 4 = 144Mbyte$ となり、メモリ上に全てのデータが載らないことになる。

計算量は $O(N^3)$ で増加する。そこで、実際の計算時間も $O(N^3)$ で増加するという推測のもとに、頂点数を増加させた場合の計算時間を比較した。この結果を図 6 に示す。

実メモリ空間にデータが収まらなくなると、オーダを無視した膨大な計算時間がかかることがわかる。実際 6000 頂点以上になると、ダイクストラの実装は見込んだ時間を大幅に過ぎても終らないため、1 頂点のみを取り出して、結果を見積もっている。例えば、8000 頂点の最短経路問題を 10PE を用いて Dijkstra のアルゴリズムで解こうとすると約 74 時間（3 日以上）かかってしまうことになる。一方、同じ条件下で Floyd のアルゴリズムを用いると、約 4 時間でこの問題を解くことが可能である。

5 おわりに

本論文では、SR2001 とその上に実装された ParallelWare の特性を考察し、その上で最短経路問題を解く二つのアルゴリズムを実装・評価した。

今回のフロイドのアルゴリズムの実装方法は全実行時間に占める通信時間の大きさが PE 数に依存しないため、ほぼ理想的なスピードアップ率を達成できた。また、問題のサイズに関しても、スケーラビリティに優れていることがわかった。

参考文献

[1] 中澤 喜三郎 , 中村 宏 , 朴 泰祐 : "超並列計算機 CP-PACS のアーキテクチャ", 情報処理学会誌 , Vol.37, No.1, pp.18-28 , Jan. 1996.

[2] *HITACHI Computers User's Guide ParallelWare -C-*, 1995.

[3] *HITACHI Computers Reference ParallelWare -C-*, 1995.

[4] Alfred V.Aho John, E.Hopcroft and Jeffrey D.Ulman, *Data Structures and Algorithms*, 1983.