ループレベルの並列処理における実行時間を考慮したスケジューリングアルゴリズムとその評価

7F−4

三浦健一†　島崎眞昭‡

(†九州大学工学部　‡九州大学大型計算機センター)

1．まえがき

並列処理の問題の中に，プログラムのどの部分をどのプロセッサにどのような順番で割り当てれば，並列計算機の性能を最大限発揮させることができるのかというスケジューリング[1]-[4]の問題がある．ループレベルでのスケジューリングアルゴリズムの代表的なものとして，静的スケジューリング，動的スケジューリング，ガイデッドセルフスケジューリング[2]（以下ＧＳＳ）がある．このうち，静的スケジューリングはコンパイル時にスケジューリングを行うために，ループ本体の実行時間にばらつきがある場合負荷の分散が不平等になる恐れがあるが，実行時のオーバーヘッドはほとんどない．逆に動的スケジューリングの場合，実行時にスケジューリングを行うために，そのオーバヘッドが問題になるが，ループ本体の実行時間にばらつきがある場合でも負荷が各プロセッサに均等に分散される．このように，実行時のオーバーヘッドと負荷の分散不均一によるオーバーヘッドはトレードオフの関係にあり，両方を同時に改善することは困難である．ＧＳＳは静的スケジューリングに比べ，多少実行時のオーバーヘッドを犠牲にすることで，負荷の均一化を図っているが，完全な負荷の均一化はできない．

本論文では，負荷分散の均一化の観点から静的動的融合スケジューリングアルゴリズムとｉｆ分岐を考慮したＧＳＳの２つのスケジューリングアルゴリズムを提案する．いずれのアルゴリズムもループ本体の実行時間が既知の場合に使用可能であり，これら２つのスケジューリングアルゴリズムを組み合わせて用いることで，従来の動的スケジューリング同様に負荷分散の均一化を図ることができ，しかも従来の動的スケジューリングに比べ大幅に実行時のオーバーヘッドを削減することができる．

2．基本的事項

2．1　ガイデッドセルフスケジューリング[2]（ＧＳＳ）

ＧＳＳは一度に複数個の繰り返しを割り当てることにより動的スケジューリングの欠点を補っている．ただし，毎回同じ個数を割り当てたのでは負荷の分散が不平等になるので，残りの繰り返しの個数に応じて割当数を決定する．ＧＳＳでは現在残っている繰り返しの個数をプロセッサの個数で割った数の繰り返しをプロセッサに割り当てていく．

現在残っている繰り返しの個数　･･･ curr_n
新しく割り当てる繰り返しの個数 ･･･ new_n
プロセッサの個数　･･･ P個

とすると

$$new\_n = \left\lceil \frac{curr\_n}{P} \right\rceil$$

個の繰り返しをプロセッサに割り当てる．

ＧＳＳは，各プロセッサの開始時間が異なっていても負荷が均等に分散されるといった特徴がある．

2．2　ループ本体の実行時間

本論文で提案するスケジューリングアルゴリズムは，ループ本体の実行時間が既知の場合に使用可能である．またループ本体中にｉｆ分岐が含まれる場合，ループ本体の実行時間が各繰り返し毎に異なるが，これらの実行時間の中で最も短いものを最速実行時間，最も長いものを最悪実行時間とする．

3．静的動的融合スケジューリングアルゴリズム

静的スケジューリングアルゴリズムと動的スケジューリングアルゴリズムを組み合わせて用い，前半部分を静的スケジューリング，後半部分を動的スケジューリングによって処理を行なうことにより，実行時のオーバーヘッドが少ないという静的スケジューリングの特徴，および負荷の均一化を図ることができるという動的スケジューリングの特徴を生かし，より高速なスケジューリングアルゴリズムを実現することができる．静的スケジューリングの割合と実行時，及び負荷の分散の不均一によるオーバーヘッドの関係を図１に示す．

実行時におけるオーバーヘッドは，静的スケジューリングの割合が少なくなる（動的スケジューリングの割合が大きくなる）に従って増大する（傾きをhとする）．

一方，負荷分散の不均一によるオーバーヘッドは，静的スケジューリングの割合を増やしていっても，あるところまでは負荷の均衡がとれていることが分かる．負荷の均衡がとれなくなる（図１のＡ）静的スケジューリングの割合は次のように求める．

繰り返しの回数　･･･N回
最速実行時間　･･･best
最悪実行時間　･･･worst
プロセッサの個数　･･･P個

とすると，各プロセッサがk回ずつ繰り返しを実行し，１つのプロセッサ（P1とする）のみk回の実行がすべて最悪実行時間で行なわれ，P-1個のプロセッサの実行がすべて最速実行時間で行なわれる場合に，残りの繰り返し(N − k * P)をP-1個のプロセッサが最速実行時間で処理したとしてもP1の終了時間より早く終了しない，つまり

$$(worst - best) * k \leq \frac{(N - k * P) * best}{P - 1}$$

を満たす最大のkを求める．これより，負荷の均衡がとれる静的スケジューリングの限界は

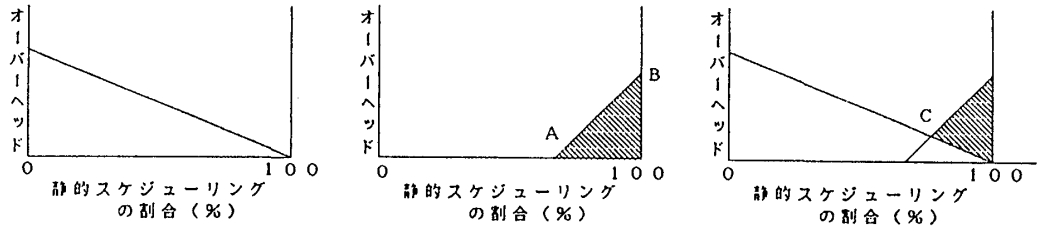
$$P * \frac{N * best}{(P - 1) * worst + best}$$

となる．

また１００％静的スケジューリングで処理した場合，各

Performance Evaluation of Scheduling Algorithms in Parallel Processing

Ken'ichi MIURA　Masaaki SHIMASAKI
(Kyushu Uviversity)



（1）実行時のオーバーヘッド　（2）負荷分散の不均一におけるオーバーヘッド　（3）（1）＋（2）

図1．実行時および負荷の不均一によるオーバーヘッド

プロセッサの終了時間は最悪の場合，
(worst − best) * N / P
開きが生じる．（図1のB）
これより図1のCは

$$P * \frac{N * best + (P-1) * N * h}{(P-1) * worst + best + P * (P-1) * h}$$

で与えられる．
静的動的融合スケジューリングアルゴリズムは，図1のCまでを静的スケジューリングで処理し，その後を動的スケジューリングで処理を行なう．

4．if分岐を考慮したGSS

GSSはある程度は負荷の分散の均一化を図ることができるが，ループ本体中にif分岐が生じる場合，負荷を各プロセッサ間で完全に均等化させることはできない．そこでループ本体中にif分岐が含まれる場合においても負荷の均一化を図ることができるif分岐を考慮したGSSについて述べる．
if分岐を考慮したGSSも静的動的融合スケジューリングアルゴリズム同様最悪の場合を考慮する．つまり，新しく繰り返しを割り当てる際に，もし新しく割り当てた繰り返しが全て最悪実行時間で処理され，割当分を引いて残った繰り返しが全て最速実行時間で処理される場合においても，新たに割り当てるプロセッサの終了が一番最後にならないように，割り当てる繰り返しの個数を設定する．
つまり
現在残っている繰り返しの個数　・・・ curr_n
新しく割り当てる繰り返しの個数　・・・ new_n
プロセッサの個数　・・・ P個
最速実行時間　・・・ best
最悪実行時間　・・・ worst
の場合

$$\frac{best * (curr\_n - new\_n)}{(P-1)} \geq worst * new\_n$$

の範囲でnew_nを決定する．これより

$$new\_n = \left\lceil \frac{best * curr\_n}{worst * (P-1) + best} \right\rceil$$

となる．
このif分岐を考慮したGSSは，開始時間が異なっても負荷の均一化を図ることができるという，GSSの特徴を生かして静的スケジューリングと組み合わせて用いることができる．

5．評価

静的動的融合スケジューリングアルゴリズム，及びif分岐を考慮したGSSの評価を表1に示す．最速実行時間と最悪実行時間の比が1：2，1：3，1：4の各場合において，プロセッサの個数を4個，8個，16個と変化させた場合の共有変数へのアクセス回数を示している．静的動的融合スケジューリングアルゴリズムで30〜60%動的スケジューリングのオーバーヘッドを削減することができる．if分岐を考慮したGSSは特に繰り返しの個数が多い場合，従来の動的スケジューリングに比べ大幅にオーバーヘッドを削減することができる．

表1．各アルゴリズムの性能評価

繰り返しの個数（オーバーヘッド） ＼ 最速：最悪		1:2			1:3			1:4		
		P=4	P=8	P=16	P=4	P=8	P=16	P=4	P=8	P=16
100	融合（前処理）	44	52	52	60	68	68	72	76	84
	if分岐を考慮したGSS	16	29	41	24	40	57	30	49	72
1000	融合（前処理）	432	472	488	600	640	664	696	728	744
	if分岐を考慮したGSS	31	59	103	45	86	150	57	109	189
10000	融合（前処理）	4288	4672	4848	6000	6368	6520	6924	7248	7392
	if分岐を考慮したGSS	46	92	172	67	135	252	86	174	326

6．まとめ

今回は，静的動的融合スケジューリングアルゴリズムとif分岐を考慮したGSSについて述べた．ループ本体の実行時間が既知であるという制約付きではあるが，従来の動的スケジューリングに比べ大幅にオーバーヘッドを軽減することができるという点で，これらのアルゴリズムは有効であると思われる．

参考文献

[1] C.D.Polychronopoulos, D.J.Kuck, D.A.Padua "Utiliizing Multidimensional Loop Parallelism on Large-Scale Parallel Processor Systems" IEEE Transactions on Computers, Vol.38, No.9, September 1989, pp.1285-1296.
[2] C.D.Polychronopolous, D.J.Kuck "Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers" IEEE Transactions on Computers, Vol.c-36, No.12, December, 1987, pp.1425-1439.
[3] Z.Fang, P.Tang, P.Yew, C.Zhu "Dynamic Processor Self-Scheduling for General Parallel Nested Loops" IEEE Transactions on Computers, Vol.39, No.7, July 1990, pp.919-929.
[4] J.Chow, W.Harrison III "Switch-Stacks: A Scheme for Microtasking Nested Parallel Loops" Supercomputing'90, November 12-16, 1990, pp.190-199.