2014 年 5 月 26 日 情報システム基盤学合同輪講発表資料

Compiler-assisted leakage energy optimization of media applications on stream architectures

著者： Shan Cao, Zhaolin Li, Zhixiang Chen, Guoyue Jiang, Shaojun Wei

出典： *14th International Symposium on Quality Electronic Design, IEEE, pp. 120 - 127, March 2013.*

発表者： 高性能コンピューティング学講座 本多研究室　1453001 石川雄介

1　はじめに

ストリーミングアーキテクチャは多数の演算器を持つことで，映像処理の性能を上げてきた．ストリーミングアーキテクチャの多くは Very Long Instruction Word(VLIW) を用いる．しかし VLIW では通常のアーキテクチャに比べて多くの演算器をもつ必要があるため，消費エネルギーが高くなってしまう．特に半導体プロセスの微細化に伴って増大しているリークエネルギーを削減することが必要である．

リークエネルギーを削減する手法の 1 つとして，アイドル状態の回路の電源をオフにすることでリーク電流を遮断するパワーゲーティング (PG) が知られている．しかし電源のオンオフの切替には余計な消費エネルギーが伴い，このオーバーヘッドと削減した消費エネルギーが釣り合う時間，BreakEven Cycle(BEC) より長くスリープをさせなければならない．そこで本論文では，演算器の PG が可能なストリーミングアーキテクチャの VLIW 命令をスケジュールすることで電源のオンオフの回数を減らす手法を提案する．

2　Imagine プロセッサ

Imagine プロセッサは，加算器 3 つ，乗算器 2 つ，除算器 1 つの 6 つの演算器によって構成されるクラスタを複数持つストリーミングアーキテクチャのプロセッサである．その構成を図 1 に示す．各クラスタは VLIW 命令によって 6 つの演算器を同時に使用することが可能である．また今回用いるものは演算器の PG を行うことができる．

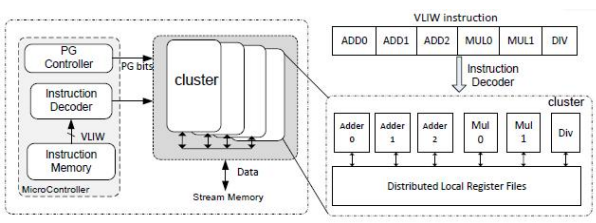

図 1: Imagine プロセッサ

3　提案スケジューリング手法

本研究ではまず VLIW 命令をスケジュールができる空間を定義する．次にその空間内でスケジューリングを行い，アイドル期間をまとめることで PG の効果を高める手法を提案する．

3.1　スケジューリング空間

演算を移行することができる空間は 3 つの観点から制約を受ける．スケジューリングではこれらの制約を満たす空間でのみ，移行を行う．

1 つ目は演算器の制約で，ある演算器を使う演算はその演算器と同じ種類の演算器に移行することはできるが，他の種類の演算器に移行することはできない．

2 つ目はハードウェアリソースの制約で，リソースの競合が起こる移行はできない．

3 つ目は時間の制約で，演算間の依存関係の解決が間に合わない時刻や，現在より終了時刻が遅れてしまう時刻への移行はできない．

移行可能な最大の範囲は絶対スケジューリング空間と呼ぶ．これは，演算を可能な限り早く行おうとする性能指向スケジューリングと，性能指向スケジューリングと最後の演算が終わる時刻は同じだが，可能な限り演算を遅くに行おうとする逆性能指向スケジューリングによって決まる．このスケジューリングは既存の技術によって可能である．

また，演算を移行できる時間は，前後の依存関係のある命令によって制約を受ける．この前後の命令によって決まる移行できる空間を，相対スケジューリング空間と呼ぶ．

これらの 3 つの制約による空間内でのみスケジューリングを行えば，プログラムが正しく動作し，実行時間が長くならないことが保証されている．

3.2　スケジューリングアルゴリズム

スケジューリングは性能指向スケジューリングを行った命令列に対して行う．このアルゴリズムは演算を使用する優先度の高い演算器に集めることで，使われなくなった演算器の PG を行いやすくする空間的なスケジューリングと，演算器のアイドル期間をまとめることで，PG が行える時間を延ばす時間的なスケジューリングを，演算の移行が行われなくなるまで続ける．命令のスケジューリングア

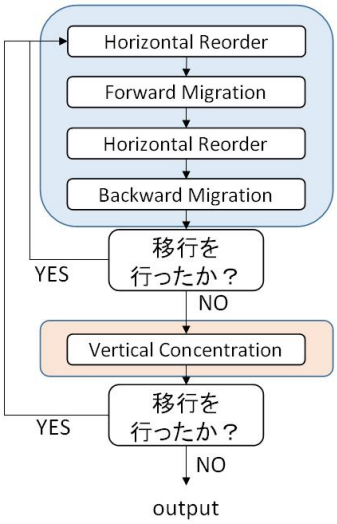

図 2: スケジューリングアルゴリズム

ルゴリズムを図 2 に示す．以下にアルゴリズムで用いられる処理の詳細を示す．

Horizontal Reorder では現段階で同時刻にスケジュールされている演算を，相対スケジューリング空間の大きいものほど優先度の低い演算器に割り当てられるように並べ替える処理である．これは以降の Migration 処理をしやすくするために行う．

Forward Migration では演算を，現在の演算器より優先度が 1 つ高い演算器に移行する．このとき，移行先で可能な限り後の時刻に演算を割り当てる．この処理は移行によって PG の効果が下がってしまうようならば行わない．

Backward Migration では Forward Migration と同様の処理を行う．ただし，移行先で可能な限り前の時刻に演算を割り当てる点が異なる．

Vertical Concentration では各演算器毎に演算が実行されないが BEC の観点から電源をオフにすべきでない nop 命令を電源をオフにできる期間に隣接させる．nop 命令を演算と交換することを連続して行い，PG の効果が上がる経路が見つかるか，全ての経路を探索するまで調べる．この処理はアルゴリズムの中で計算量が最も大きく，スケジュールを行う範囲の演算数と nop 数の積より，更に計算量が大きい．

4　評価実験

実験ではシノプシスのツールと TSMC のライブラリを使用して，VelilogHDL の合成を行った 65nm プロセスの電力モデルを使用した．比較は，C 言語の記述をコンパイルした段階の COS[1]，COS に性能指向スケジューリングの一種を適用した List[2]，提案手法である Prop の 3 つの手法に対して行う．評価指標は全アイドル期間中のスリープ期間の割合である PG efficiency，BEC を超えてスリープしたサイクル数である EPG Cycle，全スリープ期間中の EPG Cycle の割合である EPG efficiency を用いる．

PG efficiency の結果を表 1 に示す．提案手法ではほとんどのアプリケーションで 90%以上の PG efficiency を達成した．また，演算器毎に調べたところ，提案手法は加算器に対して効果が高かった．これは他の演算器に比べて演算器の数が多く，空間的なスケジューリングが行いやすかったためである．

表 1: サイクルレベルの実験結果

Kernels	PG efficiency (%)			EPG cycles			EPG efficiency (%)		
	COS	List	Prop.	COS	List	Prop.	COS	List	Prop.
FFT	56.37%	76.04%	93.75%	75	99	128	83.33%	76.04%	93.75%
DCT	75.42%	77.62%	92.87%	67	77	102	69.07%	78.57%	87.18%
Quan	54.77%	91.88%	98.14%	206	328	366	82.07%	88.65%	93.85%
IQUAN	15.91%	87.88%	87.88%	8	60	60	57.14%	100.0%	100.0%
blockscan	83.71%	97.87%	99.31%	195	253	266	84.05%	93.36%	96.73%
coefftoken	23.38%	80.36%	91.01%	25	74	103	58.14%	73.27%	84.43%
antialias	69.50%	92.29%	100.0%	65	95	109	78.31%	81.89%	87.90%
MagDetector	14.58%	42.43%	61.43%	4	12	23	57.14%	66.67%	88.46%
levelcode	54.70%	28.95%	100.0%	43	11	29	70.49%	64.71%	76.32%
runbefore	76.61%	64.44%	83.50%	75	37	48	67.57%	80.43%	88.88%

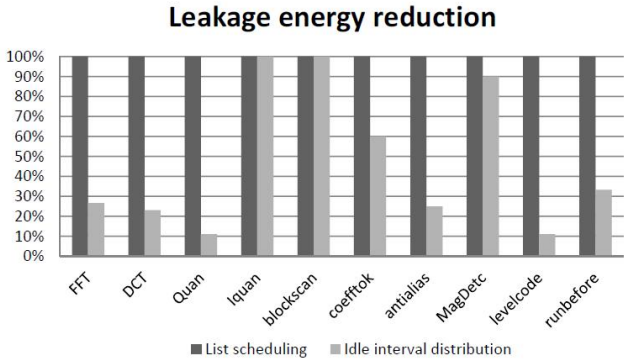

図 3: 3 つの加算器のリークエネルギー

次にサイクルレベルの結果と電力モデルから計算したリークエネルギーの比較を行った．図 3 に List と Prop について，加算器 3 つを比較した結果を示す．提案手法は，ほとんどのアプリケーションでリークエネルギーを削減でき，平均で 52%の削減となった．一方 iquan,blockscan ではリークエネルギーが削減できていないが，これは演算の移行を行える余地がほとんどなかったためである．

5　おわりに

本論文ではリークエネルギーの削減をするための，VLIW 命令列のスケジューリング方法を提案した．これにより，手法の適用前と比べて実行時間を増やさずにリークエネルギーを平均で 52%削減することに成功した．

参考文献

[1] P. Mattson, “A programming system for the Imagine media processor,” Ph.D. dissertation, Stanford University, 2002.

[2] D. Landskov, S. Davidson, B. Shriver, and P. W. Mallett, “Local microcode compaction techniques,” ACM Comput. Surv , vol. 12, no. 3, pp. 261-294, Sep. 1980.