2004－ARC－160　(7)
2004／12／2

チップ内ネットワークにおけるトポロジに対する考察

山田　裕[†]　鯉渕道紘[†]　松谷宏紀[†]
安生健一朗[††]　上樂明也[†]　天野英晴[†]

集積技術の向上により System on a Chip (SoC) に搭載できる IP コア数が増大しており, 従来利用されてきたオンチップバスでは通信部分がボトルネックとなってきている. そのため, チップ内でスイッチベースのネットワークを利用する Network on a Chip (NoC) が提案されている. 本稿では, NoC で利用されているトポロジである Mesh, Torus, H-Tree, Fat Tree, また Torus と Tree を内包するトポロジである Fat H-Tree に対し, SoC のターゲットであるストリーム処理の通信部分の性能評価を行なった. 評価対象のアプリケーションは, JPEG コーデック, SISO ビタビ復号器を用いた. 実行結果は JPEG コーデックではどのトポロジでもほぼ同等となったが, SISO ビタビ復号器では, Fat H-Tree が, H-Tree に対し, 約 20%, Mesh, Torus, Fat Tree に対し, 約 6%高い性能を示した. その結果, アプリケーションに適したトポロジを用いることで, 少ないスイッチ数でトーラスと同等の性能を得るられることが示せた.

Analysis about Topologies of Network on a Chip

Yutaka Yamada,[†] Michihiro Koibuchi,[†] Hiroki Matsutani,[†] Kenichiro Anjo,[††]
Akiya Jouraku[†] and Hideharu Amano[†]

Recent advanced semiconductor technologies allow more number of IPs in System on a Chip (SoC), but traditional on-chip bus is becoming to be the new bottleneck. Instead, various types of Network on Chip (NoC) have been proposed, but the impact of topology has not been well evaluated. In this paper, we evaluated the performance of common NoC's topology, such as Mesh, Torus, H-Tree and Fat Tree, and the hybrid topology of both Torus and Tree, called "Fat H-Tree", using the stream processing of JPEG codec and SISO Viterbi decoder. In JPEG codec, the performance of all the topology is almost equal. However, in SISO Viterbi decoder, the performance of Fat H-Tree is 20% better than that of H-Tree, and 6% better than those of the others. As a result, we show that the topology having small switches benefits a similar performance gain of Torus.

1. はじめに

集積技術の向上により, System on a Chip (SoC) の応用分野は広がり, 性能向上も著しい. また, 多くの IP コアを搭載した SoC も実現可能となってきている. 従来の AMBA[1] や CoreConnect[2] に代表されるオンチップバスでは, このような IP 数の増加による通信の増大に対応できず, ボトルネックとなることが指摘されている[3].

このような背景から, 多くの IP コアを接続しても高い通信性能を得ることができる Network on a Chip (NoC) が注目されてきている[4]. NoC では, 従来並列計算機や SAN で用いられてきたパケット転送技術をチップ内の IP コア間のデータ転送に応用し, 経路を分散することでオンチップバスの問題点を解決する.

一方, SoC の主要なターゲットである通信系, マルチメディア系のストリーム処理の特徴として, 次の二点が挙げられる.

(1) 全体の処理は複数のプロセスによりシーケンシャルに実行: それぞれのプロセスは独立に処理できるため, 全体としてパイプライン的に処理が可能である.

(2) 高い並列性: ストリーム処理を効率的に処理するためには, 負荷の高いプロセスに多くの演算リソースを割り当てる必要がある. その際に, データの分配・収集の通信が発生する.

ストリーム処理の特徴から, NoC のトポロジにはパイプライン的な処理の為の通信や並列処理の為の分配・収集の通信が効率的に処理できることが要求される. 従来の SAN や並列計算機内のネットワークと異なり, NoC ではアプリケーションに特化した設計を行なうため, トポロジもアプリケーションに適したもの

† 慶應義塾大学理工学研究科
Department of Science and Technology, Keio University
†† NEC エレクトロニクス 株式会社
NEC Electoronics Corporation

を選択することが重要となる.

二次元 Mesh などの格子構造を持つトポロジはパイプライン的な処理を行なうには効率的であるが, データの分散・収集などの集合通信において, パケットの衝突を避けることが難しい. 一方, Tree 構造を持つトポロジは分散・収集の通信には適するが, パイプライン的な処理には不向きである. そこで, 我々は格子構造, Tree 構造の特徴を持つ Fat H-Tree トポロジを提案した[5)6)].

本論文では, 一般的な NoC のトポロジである二次元 Mesh, 二次元 Torus, H-Tree, Fat Tree と, Fat H-Tree に対してストリーム処理の通信部分の性能評価を行なった.

以下では, 2 章で JPEG コーデック, SISO ビタビ復号器を例にストリーム処理の特徴について述べる. 3 章で NoC について解説する. また, 4 章で評価方式について説明し, 結果を報告する.

2. ストリーム処理

本論文では, ストリーム処理の一連の演算プロセス部それぞれを「タスク」と, タスク間に送信されるデータを「ストリーム」と呼ぶ.

ストリーム処理の一例として図 1(a), 図 1(b) に JPEG コーデックと SISO ビタビ復号器のデータフローを示す. 図中の四角形はタスクを, 矢印はストリームを表わしている.

各タスクは一定の大きさのストリームデータを受信すると, 種々の演算を施し, その結果を再びストリームとして出力する. 各タスクはストリームの入出力のみに注意すれば良く, 独立した動作が可能である.

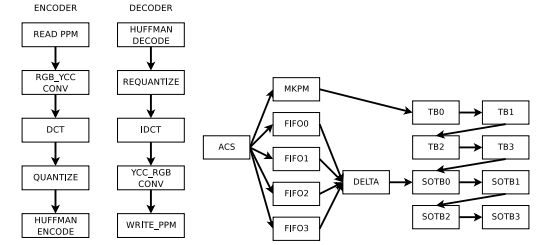
一方, ストリームの処理はストリームデータを送信元から送信先へと通信することのみを行なえば良く, タスク内部の処理とは独立して動作をすることが可能である.

マルチメディア分野, 通信分野で広く利用されているストリーム処理では, 複数のタスクがシーケンシャルに実行されるため, パイプライン的に処理をすることが容易である. また, 負荷の高いタスクに多くの演算リソースを割り当てることで, パイプラインをスムーズに実行することが可能である. ただし, その際にはストリームデータの分配・収集が必要となる.

3. Network on a Chip (NoC)

3.1 データ転送とトポロジ

従来の SoC では AMBA[1)] や Core Connect[2)] に代表されるオンチップバスで通信を行なうことが一般的である. しかし, SoC に搭載可能な IP コア数が増大するにつれ, バスがボトルネックとなることが懸念されてきている. このボトルネックを解消するために, 並列計算機や SAN などで用いられているルータスイッチベースの相互結合網をチップ内の通信に応用する方式である Network on a Chip (NoC) が提案されている[4)].



(a) JPEG　　(b) SISO ビタビ復号器

図 1　ストリーム処理のフロー

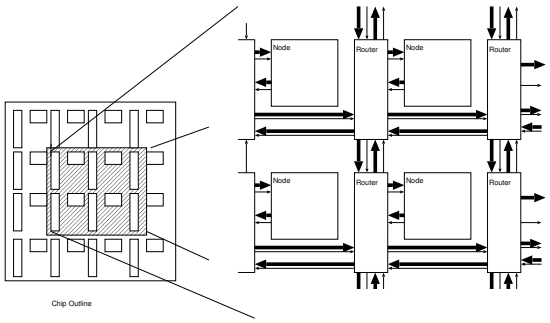


図 2　Network-on-Chips の概要

図 2 に示すように, NoC は並列計算機のノードに対応する IP コア部と, スイッチ, それらを接続するための配線から構成される. 従来の相互結合網のノードに相当する各 IP コアから送信されるデータは, アドレス情報や制御情報を含んだパケット構造を持ち, いくつかのスイッチを経由して送信先のノードへと伝送される.

古典的なオンチップバスに対する NoC の利点として, 同時に複数対のデータ転送を行なうことが可能である点が挙げられる. そのため通信の衝突の発生を抑えることができ, 高い通信性能, 及びスケーラビリティを得ることが可能である.

NoC で用いられるデータ転送方式, トポロジを以下に示す.

3.1.1 データ転送方式

- パケット方式: Guerrier らが提案した NoC である SPIN[7] では, 送信ノードでルーティング情報を付加したパケットを生成する転送方式を用いているパケット方式では, 送信先等のルーティング情報と転送データは単一の伝送路を時分割で利用する.
- アドレス並走方式: 我々の先行研究である Black-Bus[8] や QuickSilver 社の ACM[9] が用いている方式である. アドレス並走方式ではルーティング情報と転送データを同一サイクルで転送する. そのためルーティング情報を転送するための専用の通信線が必要となる.

3.1.2 トポロジ

チップ上に実装される NoC では, 利用されるトポロジは二次元構造のものが広く利用される. 以下に, NoC で利用される代表的なトポロジ (図 3) を示す.

- Mesh, Torus: Dally らや Marescaux らが提案している NoC で用いられている[4][10]. Mesh や Torus は近隣のノードとの通信距離が短かく, また, 経路分散が容易である. しかし, データの分散・収集といった通信には不向きである.
- H-Tree: QuickSilver 社の ACM[9], Caspi らの SCORE[11] で用いられている. H-Tree はチップ上に Tree 構造を効率的に配置するために利用される. Tree 構造はデータの分散・収集に優れているが, 経路の分散ができない為, 根の近辺にトラフィックが集中しがちである. また, 近隣であっても木構造の境界に存在するノード間での通信距離が大きくなることが問題として挙げられる.
- Fat Tree: SPIN で用いられている[7]. H-Tree を拡張した構造をとっており, 経路の分散が可能となっていることが特徴として挙げられる. また, 目的地まで通信距離が等しい経路が多く存在するため, 動的ルーティングを行なうことが適している. ただし, H-Tree と同じく木構造の境界に存在するノード間での通信距離が大きくなる問題がある.

3.2 NoC 上でのストリーム処理

2 章で述べたように, ストリーム処理は演算を行なうタスク部とデータ通信を行なうストリーム部にわけることができる. タスクを各ノードに割り当てることで, NoC のネットーク部でストリームの処理を行なうことが可能である. 各ノードは, 演算に必要な大きさのストリームデータを受信した後に, タスクに応じた演算を施し, ストリームデータとして送信する. ストリームデータはいくつかのスイッチを経由して送信先のノードに届けられる.

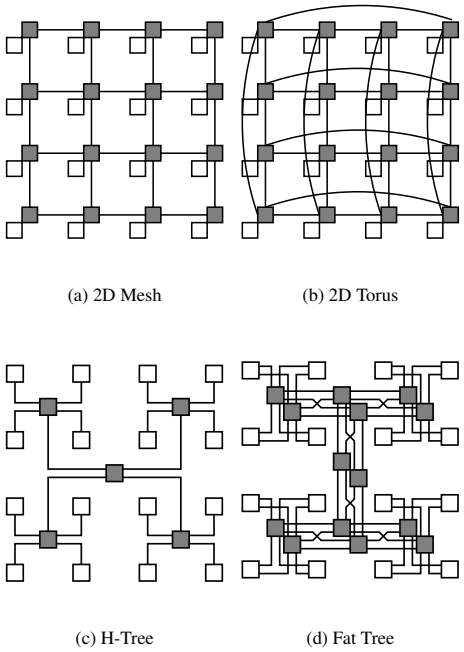


図 3 トポロジ (16 ノード)

図 4 に JPEG エンコーダを NoC に適応した場合の時間毎の各ノードの様子を示す. 縦軸は各タスク, 横軸は経過時間を示している. また, 図中の白色の四角, 灰色の四角, 黒色の四角は, ノードがそれぞれ受信, 演算, 送信の状態であることを示している.

また, 2 章で述べたようにストリーム処理を効率的に行うためには以下の機能が重要となる.

(1) パイプライン的な処理
パイプライン処理を効率的に行なうためには, 近隣のノードとの通信性能が重要となる. そのため, 近隣のノードとの通信距離が近く, かつ経路分散により経路のトラフィックを抑えることができる格子構造のトポロジが望ましい.

(2) 並列処理の効率的な実現
NoC で並列処理を効率よく行うためには, 複数のノードに対しストリームデータを分配・収集できることが重要となる. Tree 構造を持つのトポロジを利用することで, ストリームデータの分配・収集が容易になると考えられる.

以上のことから, NoC のトポロジとして格子構造と Tree 構造を合わせ持つトポロジが適していると言える.

3.3 Fat H-Tree

ストリーム処理を NoC で効率的に処理するために, 我々は Fat H-Tree トポロジを提案した[5][6]. Fat H-Tree

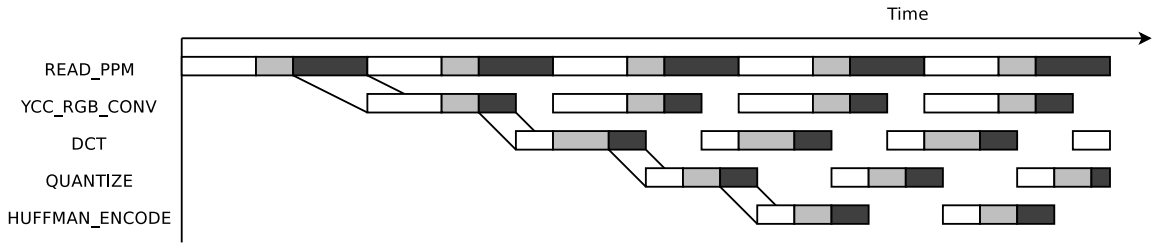


図 4 JPEG エンコーダの処理の流れ

は格子構造と Tree 構造の両方の特徴を持つトポロジである.

図 5 は, 16 ノードの Fat H-Tree を示したものである. Fat H-Tree は図に示すように, 二つの H-Tree 構造を互い違いに重ね合せた構造を取っている. また, 上辺と下辺, 及び左辺と右辺はそれぞれトポロジ的に接続している.

Fat H-Tree は二つの Tree 構造と全体として Torus 構造を内包している. これにより, H-Tree の欠点であった経路の分散ができないことによる根近辺への通信の集中や, Tree 構造の問題である境界付近での通信において転送距離が大きくなることの改善が可能である.

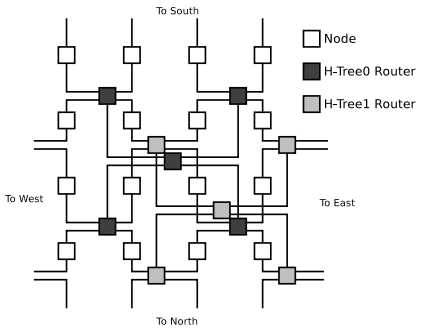


図 5 Fat H-Tree の構造 (16 ノード)

Fat H-Tree は Torus 構造を内包しているため, チップ上でのレイアウトでは長いフィードバックが必要となる. そこで Torus 構造の畳み込みを行なうことによりこの問題を解決することが可能である. 図 6 は 16 ノードの Fat H-Tree のチップ上のレイアウトを示している. このように, Torus 構造を畳み込むことで各ノードにたかだか 1 つのスイッチを置くことでレイアウトをが実現できる.

4. 評　　価

4.1 評価環境

システムレベル記述言語 SystemC で記述した BCA レベルシミュレータで通信性能の評価を行なった. こ

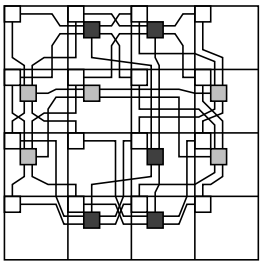


図 6 Fat H-Tree のレイアウト例 (16 ノード)

のシミュレータは, タスクを処理するための演算部, ストリーム部を処理するためのネットワーク部で構成される.

演算部は, 複数のノードモジュールで構成される. 各ノードモジュールでは, データ受信, 演算, データ送信を行なう. 今回の評価では実際の演算は行っておらず, ノードモジュールでは, データ受信後, 一定サイクル待機してデータの送信を行なっている. ネットワーク部の負荷を上げるために, 各ノードモジュールの待機時間は 1 サイクルとしている.

ネットワーク部は複数のルータモジュールで構成される. 各ルータは転送されるデータのルーティング情報により, 送信先のノードもしくはルータにデータを送信する. ルータのルーティングアルゴリズム, 接続を変更することで, 様々なトポロジに対応させることが可能である.

演算部とネットワーク部を切り分けることで, 複数のネットワークで同一のアプリケーションのシミュレーションを容易に行なうことが可能である.

また, 評価対象として, 想定するシステムは以下の通りである.

- ノード数: 16 ノード
- データ転送方式: アドレス並走方式
- 1 サイクルあたりの転送データ幅: 32bit
- 仮想チャネルは未使用

4.2 トポロジ

評価対象のトポロジとしては, 代表的な NoC のトポロジである二次元 Mesh, 二次元 Torus, H-Tree, Fat

Tree を, 格子構造, Tree 構造を内包するトポロジとして Fat H-Tree を用いた. ルーティングアルゴリズムは二次元 Mesh, 二次元 Torus では, e-cube ルーティングを用いた.

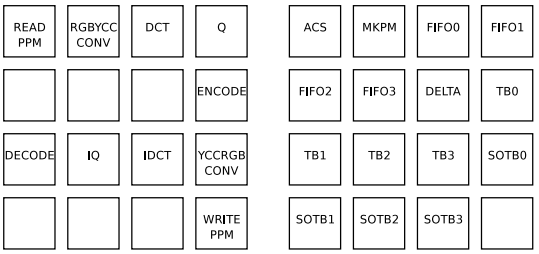
4.3 アプリケーション

評価対象のアプリケーションとして, マルチメディア系の応用例として JPEG エンコーダを, 通信系の応用例としてビタビ復号器を採用した. これらのアプリケーションのアルゴリズムを解析し, タスクのフローを作成した.

4.3.1 JPEG コーデック

JPEG は現在最も広く利用されている静止画圧縮方式である. エンコーダは 24bit の RGB カラー画像のデータを入力とし, ハフマン符号化されたデータを出力とする. また, デコータはハフマン符号化されたデータを入力とし, 24bit の RGB カラー画像のデータを出力する. 図 1(a) にタスクフローを, 図 7(a) に NoC への配置レイアウトを示す. JPEG コーデックのデータフローはシーケンシャルな構造を取っており, パイプライン処理を行なうことが容易である.

各タスクで扱われるストリームデータの大きさは 16×16 画素単位である.



(a) JPEG コーデック　　(b) SISO ビタビ復号器

図 7 レイアウト

4.3.2 SISO ビタビ復号器

SISO ビタビ復号器は無線通信規格である IEEE802.11 等で利用されている誤り訂正手法である. トレリスと呼ばれるデータ構造を持つことで最も確からしい情報を求めることができる. 通信経路から受信した 3bit×2 の情報を入力とし, 1bit の情報と 2bit の尤度を出力する. タスクのフローを図 1(b) に, NoC でのレイアウトを図 7(b) に示す. SISO ビタビ復号器のデータフローは JPEG コーデックと比較して複雑な構造となっており, シーケンシャルな通信の他に分散・収集の通信があることが特徴である.

4.4 評 価 結 果

局所性のないトラフィックの例として, 送信先をランダムに定める uniform トラフィックに関しても性能評価を行い, トポロジの静的な特性について評価をした. また, 一般的に用いられるストリーム処理の例として, JPEG コーデック, SISO ビタビ復号器の通信部分の性能を各トポロジで測定した.

4.4.1 平均ホップ数

各トポロジごとの平均ホップ数を表 1 に示す. どの通信パターンでも, Torus, Fat H-Tree が平均ホップ数が少なくなっている. これは, どちらのトポロジも Torus 構造を持ち, ノード間の平均距離が短いことが要因だと考えられる.

一方, ストリーム処理に注目すると, JPEG は完全にシーケンシャルな処理であるため Mesh, Torus, Fat H-Tree は平均ホップ数が 1 となっている.

SISO ビタビ復号器の場合は全てのトポロジにおいて JPEG に比べホップ数が大きくなっている. しかし, Mesh, Torus では SISO ビタビ復号器における平均ホップ数が JPEG に比べ 1.5 倍以上増加しているのに対し, H-Tree, Fat Tree, Fat H-Tree の平均ホップ数は約 1.4 倍と増加率が小さく抑えられている. これは, 分散収集的な通信に Tree 構造が適しているためだといえる.

表 1 ストリームアプリケーションの平均ホップ数

	uniform	JPEG	Viterbi
Mesh	2.5	1	1.83
Torus	2	1	1.5
H-Tree	2.43	1.5	2.11
Fat Tree	2.43	1.5	2.11
Fat H-Tree	2.06	1	1.39

4.4.2 シミュレーション結果

表 2 はシミュレータで通信部分の性能を測定した結果を示したものである. uniform トラフィックでは各ノードから 10000 回のデータ転送を行なった. また, JPEG コーデックは VGA(640×480) の大きさの画像を, SISO ビタビ復号器は 10000bit 分の誤り訂正を処理した場合に発生する通信を想定している.

uniform トラフィックでは経路の分散が可能な Mesh, Torus, Fat H-Tree が高い性能を示した.

一方, JPEG コーデックでは各トポロジ間で実行時間に差が見られなかった. これは, ネットワークの通信に要する時間に比べ, ノード内部のデータ通信による待ち時間や演算時間が実行時間に影響していることが原因と考えられる.

また, SISO ビタビ復号器では経路分散が可能なトポ

表 2 ストリームアプリケーションの実行時間とスループット

	スループット (flit/cycle/node)			実行時間 (clk)		
	uniform	JPEG	Viterbi	uniform	JPEG	Viterbi
Mesh	0.340	0.118	0.108	29415	811835	360097
Torus	0.368	0 118	0.108	27209	811829	360087
H-Tree	0.106	0 118	0.084	94147	811825	456075
Fat Tree	0.181	0 118	0.108	55264	811827	360085
Fat H-Tree	0.279	0.118	0.113	35853	811819	340065

ロジで高い性能を得られた. 特に, Fat H-Tree は, H-Tree に比べ 20%, Mesh, Torus, Fat Tree に対しては 6%程度高い性能を達成した.

以上の結果より, 通信に局所性の無いトラフィックパターンでは, スイッチ数や分散できる経路数の多いトポロジが高い性能を示している. しかし, トラフィックパターンが静的に解析できるストリーム処理に関しては, トラフィックパターンに適したトポロジを選択することにより, スイッチ数を減らしても十分対処することが可能である.

5. ま と め

本論文では, 今後 SoC での利用が期待される NoC に対し, トポロジの性能評価を行なった. 評価対象となるトラフィックパターンは, 局所性のないトラフィックとして uniform トラフィックを, また SoC のターゲットであるマルチメディア系, 通信系のアプリケーションとして JPEG コーデック, SISO ビタビ復号器を用いた.

uniform トラフィックでは, スイッチ数の多いトポロジが高い性能を示した. 一方シーケンシャルなトラフィックである JPEG コーデックではトポロジ間での性能にほとんど差が見られず, 分散収集的な通信を含む SISO ビタビ復号器では, Fat H-Tree のような分散収集に適したトポロジが高い通信性能を示した.

この結果, アプリケーションに特化する NoC では対象アプリケーションの通信パターンを静的に解析し, 適したトポロジを用いることにより, 少ないスイッチで実現可能なトポロジでも Torus と同等の性能を発揮できるといえる.

参 考 文 献

1) D. Flynn. AMBA Enabling Reusable On-chip Designs. *IEEE Micro*, Vol. 17, No. 4, pp. 20–27, July 1997.
2) IBM Corporation. The CoreConnect Bus Architecture, 1999. available at http://www.chips.ibm.com/products/coreconnect/index.html.
3) Luca Benini and Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, Vol. 35, No. 1, pp. 70–78, January 2002.
4) W.J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference*, pp. 684–689, June 2001.
5) 天野, 山田, 鯉渕, 上樂, 安生, 西村. リコンフィギュラブルプロセッサアレイ用チップ内結合網 Folded Fat H-tree. 第 2 回リコンフィギュラブルシステム研究会, pp. 40–45, November 2003.
6) Y. Yamada, H. Amano, M. Koibuchi, A. Jouraku, K. Anjo, and K. Nishimura. Folded Fat H-Tree: An Architecture Topology for Dynamically Reconfigurable Processor Array. *International Conference on Embedded and Ubiquitas Computing*, pp. 301–311, August 2004.
7) Pierre Guerrier and Alain Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *Proceedings of the DATE'2000 Conference*, pp. 250–256, March 2000.
8) K.Anjo, Y.Yamada, M.Koibuchi, A.Jouraku, and H.Amano. BLACK-BUS: A new data transfer technique using local address on networks-on-chips. In *Proceedings of IEEE International Parallel and Distributed Processing Symposium*, p. 10a, April 2004.
9) Bob Plunkett and John Watson. Adapt2004 ACM Architecture Overview, 2004. available at http://www.quicksilvertech.com.
10) T.Marescaux, A.Bartic, D.Verkest, S.Vernalde, and R. Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs. In *Proceedings of the Field-Programmable Logic and Applications (FPL)*, pp. 795–805, September 2002.
11) E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE). In *Proceedings of the Field-Programmable Logic and Applications (FPL)*, pp. 605–614, August 2000.