

HPCへの期待 —集積回路設計用CADの視点から—

安浦寛人

九州大学大学院総合理工学研究科

ハイパフォーマンスコンピューティングにとって集積回路設計用CADは重要な応用分野であると共に、重要な要素技術であると考えられる。応用としては、CADの中に現われる多くの組み合わせ最適化問題の効率のよい解法や各種シミュレーションの効率的な実現の問題がある。一方、CAD技術と特定用途向き集積回路(ASIC)の技術によって、比較的安価に高性能で大規模なハードウェアが実現できるようになり、HPCにおいてもハードウェアによる実現の視点が今後重要になってくると考えられる。これまでも多くの専用計算機やDSPのような専用回路が高性能計算で利用されているが、今後、ますますソフトウェアと同程度の比重でハードウェアによる計算の実現を考える時代が来ることが予想される。

Prospects for High Performance Computing - From the Standpoint of Computer Aided Design Technology for VLSI-

Hiroto Yasuura

Interdisciplinary Graduate School of Engineering Sciences
Kyushu University

For High Performance Computing, the area of CAD for VLSI is one of the important basic technologies as well as an important area of applications. CAD includes many kinds of combinational optimization problems which requires much computation time and memory resources. Various kinds of simulations in CAD are also a good application area of HPC. The progress of CAD and ASIC technologies have made it possible to implement an application specific hardware system for each computation such as special purpose computers and DSP's. The next generation of CAD which will cover system design including both software and hardware will be a fundamental technology for HPC.

1. まえがき

システム設計の目標は、高性能で高い信頼性を持つシステムを低価格で作ることにある。人間は、システム設計の各段階において、最終的なシステムの性能、信頼性、そして設計や製造さらには保守にかかるコストを考え合わせて、最適化（すくなくとも設計者はそう信じている）した設計を作りだす努力をしてきた。しかし、システムの規模が増大し、複雑度が増すにつれて、人手では設計が困難となってきた。できれば、人間が概略の設計を行ないあとは自動的に設計が行なえるような技術、あるいは人間の設計を支援する技術の確立が望まれるようになった。ソフトウェアの世界では、言語の高級化、各種プログラミングパラダイムの導入、CASE技術の確立などの努力でこのような設計の自動化や設計支援が実現されてきた。

半導体集積回路によって実現されるハードウェアの設計においても、1980年ころからソフトウェアにおける高級言語によるプログラムとのアナロジーで、シリコンコンパイラ（Silicon Compiler）というようなことが使われるようになった。従来の人手による設計の各段階をプログラムで置き換え、さらに、計算機の持つ高速な計算能力を利用して、最適化を行なわせようとするアプローチである。このような自動化技術（CAD: Computer Aided Design または DA: Design Automation と呼ばれる）は、設計の下流（物理現象に近い方）から始まった。集積回路の設計は、回路のアーキテクチャを決めるアーキテクチャ設計、アーキテクチャに従って各部分の機能を設計する機能設計、機能設計で得られた機能モジュールを論理回路として実現する論理設計、論理回路からマスクパターンデータを生成するレイアウト設計と進んでいく。1970年代から80年代の前半にかけて、レイアウトの自動化の研究と実用化が行なわれ、配置や配線問題に対して、種々のアルゴリズムが考案された。レイアウトの自動化の実用化のメドがたつと、研究の目標は上流の論理合成となった。論理合成の研究は1960年代から行なわれていたが、問題の複雑さと計算機の能力の制限から、実用とは程遠い研究が多かった。しかし、80年代の後半になって集積回路技術によって、計算機（主にワークステーション）の記憶空間と計算能力が飛躍的に増大し、論理合成が新しい設計技術として普及し始めている。その結果、回路の動作をプログラミング言語に似た言語（HDL: Hardware Design Languages と呼ばれる）で記述すれば、論理設計やレイアウト設計をシリコンコンパイラが行なって、製造プロセスに渡すマスクパターン情報を出力してくれるような状況が産み出された。

ハイパフォーマンスコンピューティング（HPC）の観点から見ると、CADやDAの技術は重要な応用分野であると共に、今後は重要な要素技術になると考えられる。応用としては、CADやDAの中に現われる多くの組み合わせ最適化問題の効率のよい解法や各種シミュレーションの効率的な実現の問題がある。一方、CAD技術と特定用途向き集積回路（ASIC: Application Specific Integrated Circuits）の技術によって、比較的安価に高性能で大規模なハードウェアが実現できるようになり、HPCにおいてもハードウェアによる実現の視点が今後重要になってくると考えられる。これまでも多くの専用計算機やDSPのような専用回路が高性能計算で利用されている。今後、設計技術と製造技術の進歩によって、ソフトウェアと同程度の比重でハードウェアによる計算の実現を考える時代が来ることが予想される。

2. 応用分野としてのCAD

昔からCADは、大規模・高速計算の応用問題の宝庫である。「大型計算機は、その次の世代の大型計算機の開発におけるシミュレーションをするために開発される」と言った笑い話も合ったほどである。ここでは、集積回路用のCADについてどのようなHPCの応用があるかを眺めてみる。

2.1 レイアウト合成

レイアウト合成は、論理回路あるいは電子回路から集積回路の製造プロセスに渡す最終的なマスクパターンを生成する段階である。現在の集積回路技術があくまでも2次元の平面上に回路を作成しなければならない制約を負っているため、回路としてそのトポロジーだけが決定されたものからジオメトリの入った世界への変換としてこの過程が必要となる。このレイアウトによって最終的な配線遅延などが決まり、回路の性能が決定される。製造プロセスの諸パラメータに依存する要素が多いので、製造プロセス側との密接な情報交換が必要である。製造メーカー側でレイアウトを行なうことも多い。しかし、FPGA(Field Programmable Gate Array)やゲートアレイなどでは、システム設計者側でレイアウトまで行なうことも増え

てきている。レイアウト結果から配線遅延などの情報を上位へフィードバックして、詳細なタイミングを考慮した論理シミュレーションなどを行なうことも多い。

レイアウト合成は、主に配置と配線の2種の処理からなる。配置は、論理素子や端子をシリコン平面のどの位置に置くかを決定するものであり、配線はそれら端子や素子の間の結線の経路を決定する作業である。配置や配線には以下のような種々の制約が設けられる。

V1. 回路はシリコン平面の長方形上に構成される。現実的に、シリコンチップは長方形であり、しかもできるだけ正方形に近いほうが加工しやすい。

V2. 配線は製造プロセスで決まる最小幅をもつ。また、チップ上のどの点においても高々定数本（通常は2～5層の配線層）の配線しか交差できない。

V3. 論理素子（ゲートやフリップフロップ）は、互いに重ならない。

V4. 素子と配線は一般的には重ならない。また、各素子や配線の間にはプロセスで決められる間隔を置かなければならない。

V5. 回路への入出力端子はチップの周上に配置される入出力端子で行なわれる。

このような制約の元で、チップの面積を最小にする配置と配線を決定するのがレイアウトの基本的な問題である。集積回路1チップあたりのコストCは大まかに言ってしまうような式で表される。

$$C = D/N + F/(YM) + T/Y + P$$

ここに、

C: 1チップあたりのコスト

M: 1ウェハ上のチップ数

D: 設計費

T: 1チップあたりのテスト費

N: 良品の数

P: パッケージのコスト

F: 1ウェハあたりの製造コスト

Y: 歩止り（良品率）

ここで、歩止りYと1ウェハ上のチップ数Mはチップ面積Aの関数となる。単位面積あたりの欠陥率をq、ウェハ面積をWとすると、

$$Y = (1 - q)^A$$

$$M = W/A$$

であるから、チップの面積、即ちレイアウトの品質がいかに大きくチップのコストに影響かがわかる。よって、特に大量に生産するものでは面積の最小化が重要な問題となる。

このレイアウトの問題は、典型的な組み合わせ最適化問題であり、NP困難な問題である。配置と配線は互いに影響し合うので、同時に双方を考慮しながら最適化するのが理想的であるが、数十万から数百万素子を含むような回路を対象とする場合には事実上不可能である。現実的には、

1) 回路の階層性を利用して、部分回路ごとにレイアウトして上位でそれをブラックボックスとして扱って統合して行く階層型レイアウト手法。

2) 概略の配置を決め、配線を行ない、あとで配置を部分的に修正する方法。

3) 回路自身の持つ規則性を利用して配置配線を決定する方法。

などが組み合わせて用いられる。階層的な手法をとることによって、1度に取り扱うデータ量（素子やモジュールの数、結線数）を小さくできる。また、2)のように配置と配線を分離することで問題の定式化も簡単になる。しかし、これらの部分問題もほとんどがNP困難な組み合わせ最適化問題となるため、これらを近似アルゴリズムやヒューリスティクスを利用して効率よく解く努力がなされている。

現実には、数万素子の回路でもワークステーションでレイアウトに数時間かかることも多く、特にFPGAのような手軽に利用できる素子ではその利用のネックになっている。

また、最近の高性能回路では、配線による遅延の問題がクローズアップされている。BiCMOSやECLの回路では、素子による遅延よりも配線遅延の方が大きい場合も報告されており、レイアウトも単に面積を最小化するだけでなく、クロック周期を決定するクリティカルパスに対応する結線の配線長を考慮する必

要が出てきている。また、クロックの分配における配線遅延によるスキューの問題などレイアウトに対する制約条件はますます厳しいものとなってきている。今後、並列処理などを取り入れたレイアウト処理の高性能化が期待されている。

2. 2 シミュレーション

集積回路の設計における設計の検証と性能評価のツールとしてシミュレーションはもっとも歴史のあるCADツールである。デバイス内部の現象を見るためのデバイスシミュレータ、電子回路レベルの回路シミュレータ（SPICE等）、トランジスタをスイッチとしてモデル化したスイッチシミュレータ、論理回路レベルの論理シミュレータ（Verilogなど）、さらにレジスタ転送レベルや動作レベルの機能シミュレータなど各種のシミュレータが実用されている。さらに、これらのレベルを縦断的に利用できる混合レベルシミュレータも開発されている。

シミュレーションの性能は、その精度と速度によって決められる。シミュレーションの速度は、対象回路の大きさとシミュレーションする対象の動作時間にも依存する。また、シミュレーションに必要な記憶容量も大きな制約条件である。精度を重視するデバイスシミュレーションや回路シミュレーションはメモリ容量と計算時間の制約から小規模な回路の短い期間の動作のシミュレーションしかできない。一方、精度は悪くなるが機能シミュレーションや論理シミュレーションでは、数十万素子のシステム全体の数万クロック程度の長い期間の動作をシミュレートできる（表1参照）。

表1. 各種シミュレーションの回路規模と速度

シミュレーションの種類	対象の回路規模	実時間との速度差
デバイスシミュレーション	1～数素子	$\times 10^9 \sim 10^{12}$
回路シミュレーション	1～数百素子	$\times 10^7 \sim 10^{10}$
論理シミュレーション	数百～数百万素子	$\times 10^4 \sim 10^8$
機能シミュレーション	数千～数百万素子	$\times 10^3 \sim 10^7$

論理シミュレーションや機能シミュレーションでは、数百万素子を含むような大規模な対象回路の数万から数十万クロック分のシミュレーションを行ないたいこともしばしばある。現在の計算機の能力では、数日を要するような場合も少なくない。専用のシミュレーションエンジンなども開発されている。また、FPGAのようなプログラマブルな素子をそのままシミュレータとして用いることも行なわれている。

論理シミュレーションのための回路への入力パターン（テストパターン）を作成する問題も重要な問題である。論理の検証のためには、少なくともすべての素子が動作するようなパターンを作る必要がある。このような入力パターンは、製造後のチップのテストにも不可欠である。このようなテスト生成問題も組み合わせ問題であり、膨大な計算時間を必要とする。設計現場の計算機の大半がこのテスト生成のために稼働しているとも言われている。

近年、テスト生成やシミュレーションをせずに検証を行なう方法として、形式的検証の技術が実用に近いレベルまで開発されてきた。論理操作を基本として、数学的に設計検証を行なう技術である。あとで触れるBDDと呼ばれるデータ構造や新しい集合演算手法の発見により、数千素子程度の回路の検証が形式的に行なわれたことが報告されている。しかし、この手法は、膨大な記憶容量と計算時間を必要とするため、より大きな計算能力の実現が切望されている。

2. 3 論理合成

論理合成とは、ハードウェアとして実現したい回路の機能記述（レジスタ転送レベルあるいは動作レベルの記述）から、自動的に論理回路を合成する技術である。研究としては1960年代から行なわれていたが、ようやくここ数年で実用的な技術となってきた。

機能記述では、各部分回路は、内部状態を含む順序機械または内部状態を含まない論理関数の形で与えられる。順序機械は、状態遷移を記述することで定義される。定義された状態のおおのに2値の符号を

割当て（状態割当て），符号の各桁をフリップフロップに割り当てることにより，順序機械から順序回路を合成する問題は，次状態関数と出力関数を計算する論理関数を実現する問題に帰着される．状態割当ては，きわめて自由度が高く，しかも最終的な回路の複雑さに大きく影響するが，回路の複雑さは簡単に見積ることが難しいので，最適化問題における目的関数の設定が難しい．

論理関数は，組み合わせ回路として実現される．組み合わせ回路の合成は，実現に使用する回路素子によってその手法が異なってくる．組み合わせ回路の合成は，論理合成の中でもっとも自動化が進んでいる分野である．

組み合わせ回路の合成問題は，以下のようにまとめることができる．

- 1) 入力： n 入力 m 出力の論理関数 $F: \{0, 1\}^n \rightarrow \{0, 1, *\}^m$ である．（*は，Don't Care）
- 2) 制約条件：使用できる素子，面積（素子数）の上限，形状，遅延の上限などがある．
- 3) 最適化の目標：面積最小の回路，遅延時間最小の回路あるいは目的関数が面積と遅延時間の関数として与えられる場合もある．一般に面積は，レイアウトをしてみないと決定できないので，面積との相関の強い素子数が中間目標として設定されることが多い．遅延時間もレイアウトに依存して決まる配線の遅延を考慮しなければならないが，これも回路の段数（入力から出力に至る経路上の素子の数の最大値）で代用することが多い．

組み合わせ回路の合成も代表的な組み合わせ最適化問題である．通常，問題を簡単にするためにいくつかのステップに分けて問題が解かれる．最初，二段実現と呼ばれる積和形の最小表現を求め，それから制約を満たすように多段回路へと変換されることが多い．

任意の1出力論理関数は，論理積の和の形式，すなわち積和表現によって表すことができる． n 入力 m 出力の論理関数を論理積項を共有した形でそれぞれの出力ごとに積和表現すると，論理和と論理積による二段実現された回路が得られる．このとき，積和表現に現われる異なる積項の数を最小化することとなる．すなわち，積項数が最小となる積和表現をみつける論理式の最小化である．論理式の最小化は，もっとも基本的なNP困難問題の一つであり，過去30年以上にわたりさまざまなアルゴリズムが提案されてきた．ここ10年間に，この分野のアルゴリズムやプログラムの開発が活発に行なわれ，現在，入力数が20程度ならば真の最適解が，50から100程度の関数については準最適解が実用的な時間内で得られるようなプログラムが開発されている．

通常の論理素子を用いた論理回路の設計では，各論理ゲートの入次数に制限があり，二段実現はできない．すなわち，回路は多段のネットワークとしなければならない．多段論理設計では，最終的なプロセステクノロジー（バイポーラ，CMOSなど）に依存しないで，できるだけ一般的な素子（AND, OR, NOT）を用いて回路を構成し，さらにテクノロジーにあわせて素子を割り当てるテクノロジーマッピングを行なう方法が良く採られる．

与えられた論理関数の論理式あるいは真理値表から一気に多段論理回路を組み上げるのはなかなか難しい．そこで，一端，二段論理の最小化を行なってこれを出発点に多段化を行なう方法が採られる．基本的な手法は，まず二段論理回路を大局的に見て多段化を行なう．この時，中間的に生成される出力をできるだけ多くのところで利用するようにして，素子数の最小化を図る．次に，回路の各部を局所的に最適化する．ここでは，回路内部に潜在するDon't Care条件を利用したり，回路の置き換えを行なったりする．ルールベースシステムのような知識処理的な手法も有効である．最後に，回路の段数を遅延条件を満たすように修正する．

論理合成のプログラムは組み合わせ回路の合成を中心に実用化されている．数千素子以上の人手設計より優れた回路を生成する合成システムも存在する．問題は，論理設計の品質から設計時間や記憶容量へと移行している．

このような論理合成の各段階で行なわれる論理関数処理に，近年2分決定木BDD（Binary Decision Diagram）と呼ばれる新しいデータ構造が導入された．BDDは，論理式に比べ実用的な論理関数を比較的小さく表すことができる．また，変数の順序を決めると最小形が一意に決まる特長を持っている．BDDの処理系やその性質に関する研究が進み，数千素子程度の実用的な回路の各素子の出力の論理関数をBDDで表現することができるようになった．BDDは，基本的に出入次数2の有向グラフであるが，数百万節点を含むようなBDDもしばしば現われる．この分野でもっとも記憶領域を消費する問題である．

3. HPCのためのCAD技術

CADは、HPCの応用分野であると共に、HPCのための基礎技術でもある。ソフトウェアの規模や複雑さはますます増大し、オペレーティングシステムや多くの大規模なアプリケーションはその保守だけでも極めて困難な作業となっている。このため、新たに開発されるハードウェアはソフトウェアに対するインターフェースを変えることがないという条件を必然的に課せられている。オペレーティングシステムやアプリケーションの方がハードウェアよりもライフサイクルが長いわけである。これは、ソフトウェアは柔らかくハードウェアは堅いという語源と矛盾している。近年のCAD技術の進歩はハードウェア自身をより柔らかくしている。また、ハードウェアをふんだんに使い、その構成を柔軟に変えられる並列処理システムを設計することを考えると、ソフトウェアの設計とハードウェアの設計を分離して考えることは難しくなる。このような状況の元で、新しいハードウェア設計手法さらにはソフトウェアとハードウェアを一体としたシステム設計手法の確立がCADの分野で重要な課題となっている。

3. 1 並列計算機アーキテクチャ設計ワークベンチ

高性能な並列処理システムを構築するためには、設計早期の段階(仕様決定や設計時)から性能評価を行い、ソフトウェアとハードウェアの設計を統合的に行ないながらシステムを最適化していく必要がある。しかし、並列処理システムの設計初期の性能評価など含めた総合的なCADシステムを構築する上で必要な技術に関してはまだ研究段階であり、問題の定式化も固まっていない。我々は、性能評価なども含めた今後のCADシステムに必要な基礎技術の開発を行うことを目的として並列計算機アーキテクチャ設計ワークベンチの構築を行なっている。特に以下の点が重要である。

- 1) どのように並列処理システム(ハードウェアとソフトウェア)を記述し、設計を行うか。
- 2) どのように並列処理システムの性能評価を行うか。
- 3) どのようにハードウェア/ソフトウェアの協調設計(Hardware/Software Co-Design)を行うか。

最終目標としては一般の“並列処理システム”に対する総合的な設計支援環境の開発であるが、最初から並列処理システムすべてをターゲットとすることは難しい。そこで、並列処理システムの1形態のスーパースカラプロセッサにターゲットを限定した。スーパースカラプロセッサを選んだ理由は、1プロセッサで並列処理が完結しており、しかも並列処理システムの持つ開発上の難しさを種々備えていると考えられるからである。

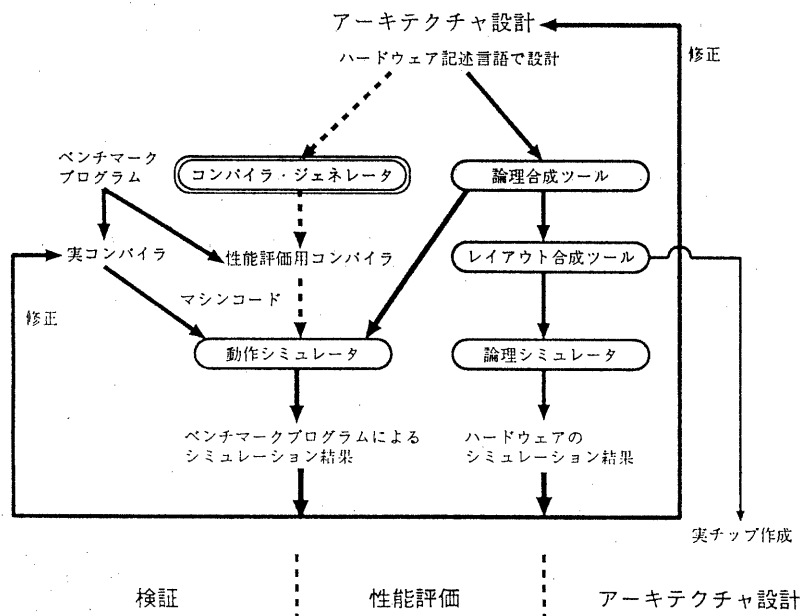


図1. 並列計算機アーキテクチャ設計ワークベンチの構成

論理合成ツール、シミュレータ、コンパイラについては、それぞれがほぼ完成された技術であり、既存のCADツールを利用することが可能である。今現在、ほとんど開発されていない部分はコンパイラを生成する部分である。ワークベンチでは、設計者がHDLによって設計を行い、その記述からコンパイラを生成するための情報を自動抽出し、その情報からコンパイラを自動生成する。生成したコンパイラで対象とする計算機システムで使用するプログラムをマシンコードにコンパイルする。HDLからシミュレータモデルを作成し、コンパイラから得られたマシンコードによるシミュレーションを行う。一方でHDLから論理合成やレイアウト合成をすることで論理/レイアウトデータを得ることができる。このレイアウトデータをもとに配線遅延などを考慮した論理シミュレーションを行って、クロック周波数などの基礎的な情報を得られる。設計者はこれらの結果から総合的な性能評価を行い、仕様を満たすまで修正を行う。特に本ワークベンチでは、コンパイラの生成を行うことが特徴となっている。

3. 2 新しいシステム設計手法にむけて

計算機アーキテクチャは、ハードウェアと基本ソフトウェア（オペレーティングシステムやコンパイラ）、基本ソフトウェアと応用プログラムの間のインターフェースである。その昔、大型機の全盛時代は、基本ソフトウェアと応用プログラムの間のインターフェースだけが一般に公開され、ハードウェアと基本ソフトウェアの間のインターフェースは計算機メーカーの内部機密であった。その後、マイクロプロセッサの登場により、ハードウェア設計と基本ソフトウェアの設計は分離され、MS-DOSのような基本ソフトウェアが現われた。さらに、UNIX文化の普及で、基本ソフトウェアと応用プログラムの間のインターフェースはかなり固定化された。これに対してRISCの登場で、ハードウェアと基本ソフトウェアのインターフェースは逆に柔軟になり、両者のインターフェースを考慮してハードウェアを設計する手法が定着した。しかし、前の大型機時代とは違い、ハードウェアと基本ソフトウェアのインターフェースは公開されている。今後、並列処理の導入に伴い、ハードウェアと基本ソフトウェアのインターフェースは再び大型機時代の様に非公開となる可能性もある。しかし、ハードウェア設計技術の進歩が全く別の道を可能としていると思われる。

新しい時代の計算機アーキテクチャの在り方としてハードウェアも基本ソフトもすべてを公開するという方向が考えられる。即ち、

System = Open Hardware + Open Basic Software + Application Software

と考えるのである。公開ハードウェアや公開基本ソフトウェアは単にそのインターフェースを公開するだけでなくハードウェア設計の完全な記述、ソフトウェアのソースコードの公開も行なう。このことにより図2のようなシステム設計が可能となる。

- 1) 公開ハードウェアと公開基本ソフトウェアの上でシステムの設計を行ないシステムを応用プログラムとして実現する。公開ハードウェアと公開基本ソフトウェアは最新の技術で作られており、十分な性能が期待できる。この段階では、従来のソフトウェア開発手法がそのまま利用できる。
- 2) システムの機能や目標性能が十分に固まった段階で、公開ハードウェアと公開基本ソフトウェアの公開情報を使って設計を完全に展開する。その上で、コスト、性能、信頼性等の目標関数に従ってシステムの最適化を行なう。この作業はハードウェアとソフトウェアの境界をどのように設定するかと言う問題として定式化できる。
- 3) 再設計の結果に基づき応用指向専用ハードウェアを合成し、その上に応用指向専用ソフトウェアをのせて最終的なシステムとする。

この考え方の基本は、

System = Open Hardware + Open Basic Software + Application Software
= Application Specific Hardware + Application Specific Software

という等式に基づいて設計の変換を行なうことにある。

このような、設計手法が成立するためには、

- 1) ハードウェアおよび基本ソフトウェアの情報の完全な公開
- 2) ハードウェアとソフトウェア共通の記述法の確立
- 3) ハードウェア／ソフトウェアのトレードオフを考慮した再設計技術
- 4) ハードウェアの完全な自動合成とコストの低い実現技術
- 5) 専用ハードウェアに向けた可変コンパイラの技術

が必要となる。それぞれの技術は現在かなり充実してきたものもあれば、まだ研究の緒についたばかりのものもある。

- 1) ハードウェアおよび基本ソフトウェアの情報の完全な公開

これは、近年問題となっている知的所有権の過剰な権利主張に対する対抗手段とも考えることが出来る。人類の共有資産としての知的創造物を積極的に発明者が公開することで、新しい価値体系を創造することにもつながる。

- 2) ハードウェアとソフトウェア共通の記述法の確立

ハードウェアとソフトウェアの双方の設計に利用できるセマンティクスとシンタクスを備えた言語が必要となる。現在のHDLやプログラミング言語の拡張では不十分である。

- 3) ハードウェア／ソフトウェアのトレードオフを考慮した再設計技術

ソフトウェアから並列化できる部分を抽出する技術やハードウェアとソフトウェアのトレードオフを評価する技術、ハードウェアの中で不要な部分を抽出する技術などが必要となる。さらには、ソフトウェア向きのアルゴリズムと対応するハードウェアアルゴリズムのデータベースを構築し、その変換を支援することも考えられる。

- 4) ハードウェアの完全な自動合成とコストの低い実現技術

自動合成技術としては、論理合成やレイアウト合成が既に実用化の域に達している。さらに、機能設計レベルの自動合成技術が充実しつつある。ハードウェアの実現も電子ビーム直接露光によるスタンダードセル方式、ゲートアレイ、FPGAなど選択の幅が大きく広がってきている。

- 5) 専用ハードウェアに向けた可変コンパイラの技術

ハードウェア設計が自由に変更できるのでそれぞれの設計に合わせてソフトウェア部分をコンパイルできるコンパイラの生成技術が必要となる。さらに、ハードウェアの設計記述からコンパイラ生成に必要な情報を抽出する技術の確立も課題である。

4. あとがき

ハイパフォーマンスコンピューティング研究会の発足にあたり、集積回路設計用CADの立場から当研究会への期待を込めて、両分野の深いかわりについて私見を述べさせていただいた。

CADの中には、HPCのさらなる発展によって広がると予想される分野が数多くある。アルゴリズム理論や計算の複雑さの理論の構築にCADの分野の実用的な問題へのニーズが大きな役割を演じたことは記憶に新しい。今後の並列計算を中心とする新しいHPCの発展にもこの分野のニーズが役立つことを期待している。

また、集積回路設計用CAD技術は、それ自身がHPCの要素技術になっている。CAD技術やそれをベースにしたASIC技術はよりシステム設計者よりに変化している。FPGAのようなハードウェアとソフトウェアの中間を行くような技術も出てきた。今後のHPCがこれらの成果を十分に活かして発展していくことを期待する。