2. 新しいデザインバランス

4 コンパイラとマイクロプロセッサ

西山 博泰
日立製作所 システム開発研究所
nisiyama@sdl.hitachi.co.jp

コンパイラとマイクロプロセッサのかかわり

マイクロプロセッサは，組み込み用途からスーパーコンピュータに到る幅広い分野で利用されている．このうち，ハイエンドシステムに用いられるマイクロプロセッサの，近年における性能向上には目を見張るものがある．たとえば，**図-1** はスーパーコンピュータの性能ランキング[1]の上位に位置するシステムの動向を示したグラフである．1993年時点ではマイクロプロセッサベースのスカラシステムの割合は約26%であったのに対して，2005年現在ではほぼすべて（約96%）がマイクロプロセッサベースのシステムとなっている．

初期の計算機システムでは機械語あるいはアセンブラによるプログラム開発が基本であった．1950年代に最初のコンパイラが開発されて以来，一部の特殊なケースを除いて，プログラム開発にコンパイラは不可欠な存在となっている．

以前のコンパイラ技術の牽引役はベクトルスーパーコンピュータに代表される高性能なハイエンドシステムであったが，現在ではコモディティ向けマイクロプロセッサがコンパイラ進化の原動力となっている．一方，RISCプロセッサの普及以降，マイクロプロセッサの性能を引き出すためには，コンパイラ最適化を考慮した総合的なアーキテクチャ設計が不可欠となっている．すなわち，マイクロプロセッサとコンパイラ技術は計算機システムの性能を向上させるための両輪を成している．

従来のマイクロプロセッサベースのシステムでは，プログラムをいかに高速に実行できるか，あるいは，使用メモリ量をいかに削減できるかが重要であった．さらに近年では，Javaなどの新たなプログラミング言語の普及，プロセス微細化に伴う電力消費の拡大，コンピュータウィルスなどセキュリティ上の脅威の増大など，従来にない新たな課題が生まれている．

本稿では，このような現状をふまえ，最適化を中心としたコンパイラとマイクロプロセッサの関係について述

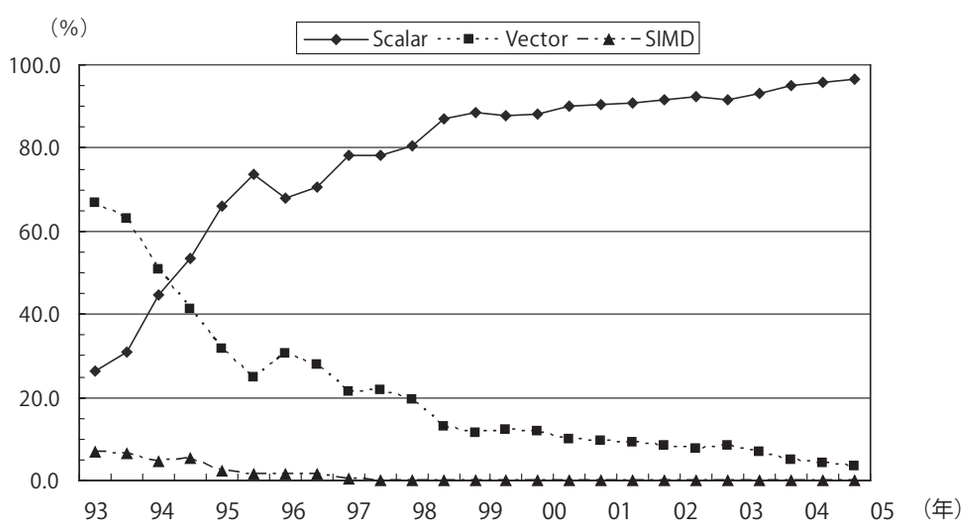


図-1　スーパーコンピュータシステムの方式内訳

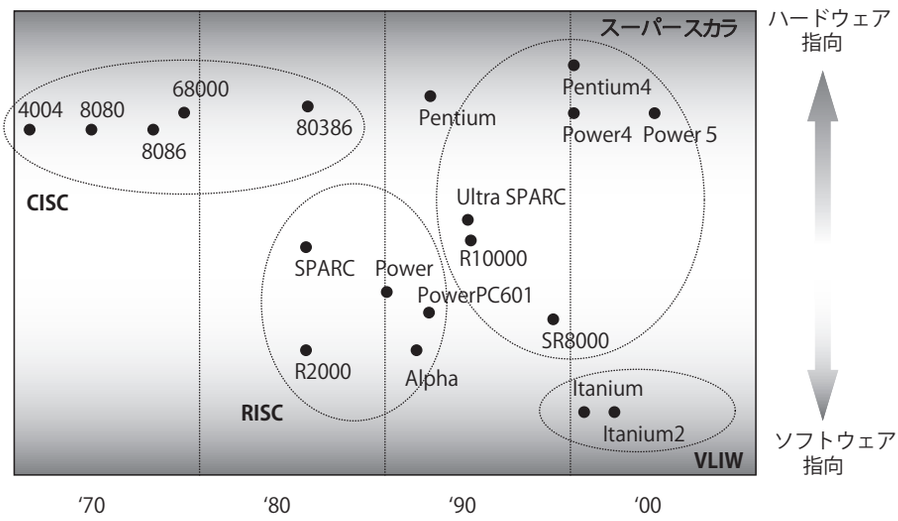


図-2　マイクロプロセッサの方式の変遷

アーキテクチャ	最適化技術		
CISC	（ダイナミックプログラミングによる）命令選択， ピープホール最適化		
RISC	レジスタ割り当て （グラフ彩色）	局所命令スケジューリング	
VLIW		大域命令スケジューリング， ソフトウェアパイプライニング， 必要レジスタ数を考慮した 命令スケジューリング	プレディケーション， 投機的命令実行
スーパースカラ			プロセッサ内部資源 利用の均等化
共通基盤技術	データ・制御フロー解析，依存解析，別名解析， 伝統的最適化（コピー伝播，共通式削除，不変式移動，演算強度軽減など）， 手続きインライン展開，SIMD 化， ループ変換，キャッシュ向け最適化（ブロッキング，パディングなど）， データプリフェッチ・プリロード		

表-1　アーキテクチャとコンパイラ最適化技術

べるとともに，動的言語，消費電力，セキュリティなど新たな話題についても取り上げる．

シングルスレッド向け最適化技術

これまでのマイクロプロセッサ向けコンパイラの最適化は，マイクロプロセッサの方式の変遷（**図-2**）に沿って，主にシングルスレッド，すなわち，逐次的な処理の高速化を中心に発展を遂げてきた．

シングルスレッド向け最適化としては，共通式削除やコピー伝播などの伝統的最適化と呼ばれるコード最適化，最適化を支える種々の解析などさまざまな技法が存在する（**表-1**）[2), 3)]．ここでは紙面の都合上，マイクロプロセッサに密着したコンパイラ最適化として，命令スケジューリングを中心とした最適化，記憶階層を意識した最適化，SIMD 命令向け最適化を取り上げる．

■ 命令スケジューリング

初期の CISC（Complex Instruction Set Computer）方式のマイクロプロセッサは，実行性能が低く，メモリも比較的高コストであったため，高機能の命令を使って命令サイズを縮小するアプローチが有効であった．コンパイラ技術という観点から見ると，CISC 全盛の当時はダイナミックプログラミングによる最適命令選択といった技術が十分に成熟しておらず，高機能な命令を使いこなすことが容易ではなかった．

このような状況の下で，CISC に対するアンチテーゼとして提唱されたのが RISC（Reduced Instruction Set Computer）プロセッサである．RISC は比較的単純な命

令をハードウェアで直接実装し，高クロックを実現した．RISCは，命令実行に際して空きサイクル（パイプラインストール）を発生させないよう，コンパイラが命令の並べ替え（スケジューリング）を行うことを期待している．

初期の命令スケジューリングは，制御フローの出入りのない連続した命令列（基本ブロック）に適用対象を限定していた．ところが，パイプライン段数の増加やスーパースカラ方式の採用などにより，複数命令を同時実行することが可能になると，基本ブロック内だけでは十分な並列性を得ることが困難となった．そこで，より広い範囲の命令を対象として命令スケジューリングを適用する必要性が生じた．

現在マイクロプロセッサ向けに利用されている命令スケジューリング技術は，RISCの勃興期とほぼ時を同じくして開発されたVLIW（Very Long Instruction Word）アーキテクチャに多くを負っている．VLIWはその名の通り命令長が長く，1つの命令で複数の演算を駆動することができる．VLIWではコンパイラによる命令レベル並列性の抽出を想定しているため，命令レベルの最適化技術の進歩に多大な寄与をした．

VLIWを源とする命令スケジューリング技術に，ソフトウェアパイプライニングとトレースベースの大域命令スケジューリングがある．

ソフトウェアパイプライニングはループの隣接した繰り返しがオーバーラップされるよう命令の並べ替えを行うことによって，命令レベルの並列性を抽出する命令スケジューリング技術である．ソフトウェアパイプライニングによってループを最適に近い性能で実行できるため，科学技術向け計算機では必須の技術となっている．ソフトウェアパイプライニング手法としては，モジュロスケジューリングという技法が広く利用されている．モジュロスケジューリングは分岐を含むループへの適用が困難であったが，その弱点を克服したエンハンスドモジュロスケジューリングと呼ばれる拡張なども開発されている．

大域命令スケジューリングは基本ブロックを越える範囲から命令を移動して命令並列性を抽出する技術である．大域命令スケジューリングはさまざまな手法が提案されているが，初期のVLIW向けに開発されたトレーススケジューリングから派生したスーパーブロックスケジューリングが著名である．

トレーススケジューリングでは実行頻度の高いプログラム中のパスをトレースと呼び，その中で命令を移動する．トレースはそれを構成する命令途中への飛び込み，および，途中からの脱出経路を含むため，命令移動に伴ってトレースに含まれない命令実行が矛盾しないよう補償コードを生成する．スーパーブロックスケジューリングはトレーススケジューリングを拡張した手法で，トレースへの入口が1カ所のみとなるよう命令を複写したものをスーパーブロックと呼び，これを命令移動の対象とする．トレースと比較すると，スーパーブロックでは命令移動に伴う補償コードの生成を簡略化することができる．

VLIWが演算の並列実行を直接制御するのに対し，スーパースカラ方式のRISCでは並列実行可能な命令をハードウェアが検出して実行する．コンパイラの観点から見ると，初期のRISCの内部実行機構はVLIWと大きく変わらず，与えられた並び順で命令発行を行う方式（インオーダ）であったため，命令スケジューリングはVLIWと同様に扱うことができる．

その後，マイクロプロセッサには実行可能となった命令から順に実行を開始するアウトオブオーダ方式が導入され，これが現在の主流となっている．アウトオブオーダプロセッサでは，プロセッサ内部に多数の命令キューやリネームレジスタを備えるなど，論理アーキテクチャと物理アーキテクチャが乖離している．これは，アーキテクチャ移行コストの増大によって，新しい命令セットを普及させる困難さが増しているため，バイナリ互換性を犠牲にしないで性能を向上することが求められていることによる．

論理／物理アーキテクチャの乖離により，最近のプロセッサではコンパイラが実際のプロセッサの動作を予測することが容易でなくなっている．これは，命令スケジューリングに際し，コンパイラがプロセッサの内部資源の利用過程をシミュレートする必要があるのに対して，アウトオブオーダプロセッサでは，キャッシュのヒット率など動的な現象に影響を受け，同一の命令列であってもプロセッサ内部の命令実行過程が大きく変化してしまうことによる．したがって，コンパイラによって事前にプロセッサ内部動作の正確な予測を行うことは困難であり，現在のところは単純化したモデルに基づいてプロセッサの動作を推測している．

■ 記憶階層向け最適化

マイクロプロセッサの記憶階層は，一般にレジスタ，キャッシュ，主記憶から構成される．アクセスサイクルと容量はこの順で増加するため，高速な記憶階層をどれだけ有効に活用できるかがプログラム高速化の鍵となる．

一般的なRISCは数十個の論理レジスタを備えている．これを活用するためのレジスタ割り当てアルゴリズムとしてはグラフ彩色による手法が確立している．ただし，プロセッサの高周波数化に伴う演算サイクルの増加や命令並列度の向上によって，プログラムが必要とするレジスタ数は増加している．したがって，命令移動に伴う必要レジスタの増減の考慮，スーパースカラ向け最適化処理におけるリネームレジスタなどプロセッサ内資源の動

きの考慮など，命令スケジューリングとレジスタ割り当ての連携が最近のプロセッサでは重要となっている．

初期のマイクロプロセッサはキャッシュメモリを備えていなかったが，マイクロプロセッサへの性能要求が高まるにつれ，キャッシュが採用されるとともにその容量も年々増加を続けている．また主記憶とマイクロプロセッサの性能の隔たりが大きくなるにつれ，キャッシュを意識した最適化の重要性が増している．

キャッシュ向け最適化としては，ループ交換などユニモジュラ変換と呼ばれるループ構造変換，ループ融合やタイリングなどキャッシュの再利用性を向上するための最適化がキャッシュベースのマイクロプロセッサに対して有効である．

また，最近のプロセッサではアウトオブオーダ実行により複数のメモリ参照を同時に処理することが可能である．これを，メモリパラレリズムと呼び，複数のキャッシュミスが同時に処理されるようプログラムを変換することで，キャッシュミスサイクルを隠蔽する手法も提案されている．

科学技術計算やマルチメディア処理では，キャッシュ再利用性の低い処理が多く，キャッシュが有効に動作しない．そこで，メモリ参照サイクルを隠蔽するために，主記憶からキャッシュへ先行的にデータを転送するデータプリフェッチが各種プロセッサで採用されている．データプリフェッチ手法としては，ソフトウェア，ハードウェア，および，その併用手法が数多く研究されている．

キャッシュメモリは数ワードから十数ワードのラインを単位として構成されるため，離散的なデータ参照に対してもライン全体のデータを転送せねばならず効率が低下する．この点は，離散的な参照を効率的に行う機構を備えたベクトル計算機に対し，後塵を拝する点の1つとなっている．これに対して擬似ベクトルと呼ばれる手法を導入し，離散参照性能を向上する方式が提案・商用化されている．

■ SIMD 命令向け最適化

最近のマイクロプロセッサは，1つのレジスタに複数（通常2～4程度）のデータを格納し複数の演算を同時に実行する SIMD（Single Instruction Multiple Data）命令を備えている．SIMD 命令は科学技術計算やマルチメディア処理の高速化に高い効果がある．

現在のところ，SIMD 命令の利用は命令に対応した組み込み関数を呼び出すようユーザが記述するケースが多い．ただし，SIMD 命令をベクトル長の短いベクトル命令とみなすことで，ベクトル化コンパイラの技術を適用することができる．また，複数の独立した演算を統合して SIMD 命令に変換する手法なども研究されている．

整数型データを対象とする SIMD 命令を自動生成する場合，SIMD 命令化によって演算精度が不足しないように保証する必要がある．このために，各演算に対して必要となるワード中のビット長を解析によって求める技術も研究されている．

マルチスレッド向け最適化技術

初期のマイクロプロセッサはシングルプロセッサで利用されることを想定して設計されていた．マイクロプロセッサの性能が向上し，ワークステーション等がマイクロプロセッサで構成されるようになると，マイクロプロセッサでマルチプロセッサを構成することが一般的となった．さらに，現在ではコンシューマ向け PC に利用されるマイクロプロセッサでもマルチプロセッサが普及を始めている．

■ 科学技術計算向け並列化

マイクロプロセッサをベースとした科学技術計算向け並列システムとしては，SMP（Symmetric Multiple Processor）や NUMA（Non-Uniform Memory Access）のような主記憶共有型のシステムと，MPP（Massively Parallel Processing）のような分散メモリ型のシステムが存在する．主記憶共有型のシステムはハイエンドサーバを中心として広く普及している．また，分散メモリ型のシステムはプログラミングの難しさから一時下火となっていたが，クラスタ型システム等の普及により見直しの機運が高まっている．

こういったマルチプロセッサシステムを対象としたコンパイラ最適化は，主に科学技術計算を対象とした自動並列化が一般に利用されている．また，OpenMP などユーザ指示による並列化手法も広く普及している．このような自動並列化ではループに着目し，ループの各繰り返しを別々のプロセッサに割り当てて並列に処理を行うことが一般的である．ループ並列化では，プロセッサ間で高速に同期を取るためにバリア型の同期を利用する．いくつかのマイクロプロセッサでは，ハードウェアで高速に同期を行う機構を備えている．ただし，ハードウェアサポートがある場合でも同期オペレーションは重い処理であるため，コンパイラ解析による不要な同期の除去，同期点を越えた命令移動などが有効である．より広い範囲で手続き間にわたって並列化を行うシステムも実用の域に入っている．

分散メモリシステム向け自動並列化技術は，HPF など分散メモリ向け拡張言語を中心に研究が行われているが，データの効果的な分散メモリへの配置や通信オーバーヘッドの削減が難しく，広範な問題に適用するにはさらなる研究の推進が必要である．

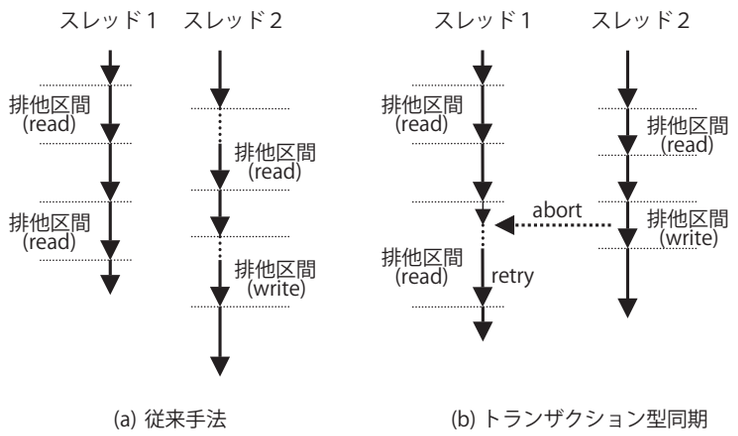


図-3　トランザクション型同期処理

■ 非数値計算向け同期高速化

科学技術計算を除くと自動並列化が適用可能な分野は残念ながらそれほど多くない．したがって，Java のような最近の言語では，プログラマによる明示的なマルチスレッドプログラムの記述をサポートしている．これらの言語ではモニタ型の同期プリミティブが利用されることが一般的である．

こういった言語で提供されるクラスライブラリはマルチスレッド環境で利用されることを想定して実装する必要があるため，実際に競合が起きないケースでも同期オーバーヘッドが生じるという問題がある．そこで，同期処理の静的な解析による冗長な同期の削除や，スレッド間で競合が発生した場合に必要となる待ち行列などのデータ構造の操作を実際に競合が発生するまで遅延させる最適化が研究されている．後者は，一般のプログラムでは競合が低い確率でしか発生しないことに着目したものである．

より積極的な手法として，データベースにおけるトランザクションと同様の考えを利用する最適化手法がある．**図 -3** にトランザクション型同期最適化の概念を示す．従来手法では排他区間（クリティカルセクション）を逐次的に実行する必要があったが，トランザクション型同期最適化ではスレッド間の共有状態の変更を伴わない区間については同時実行を許し，変更が検出された場合には実行を破棄（abort）して，逐次的に再実行（retry）を行う．ソフトウェアのみによるトランザクション型同期最適化には限界があるが，プロセッサによりトランザクション型の同期の高速化をサポートする手法も提案されている．

動的言語向けコンパイラ

比較的最近登場した Java のような言語は，プログラムの可搬性を実現するため，バイトコードインタープリタによる実行を採用している[4]．インタープリタによる実行は，機械語の直接実行と比較して，プログラムの実行性能を低下させる．そこで，実行性能の低速性を補うため，JIT（Just-In-Time）コンパイラあるいは動的コンパイラ☆1 と呼ばれる技術により，プログラムの実行中に動的にコードを生成する手法が採用されている．

Java 向けのコンパイラは FORTRAN や C++ といった従来の言語と比較するとさまざまな違いがある．特にプロセッサアーキテクチャに関連した特徴を以下に挙げる．

（a）コンパイル時間に制約がある
（b）動的にプログラムの生成／変更を行う
（c）システムの安全性を保証するため，配列添字など種々の検査を動的に行う

（a）のコンパイル時間に関しては，コンパイルに要する時間と生成される機械語の品質のトレードオフが重要となる．たとえば，従来の静的コンパイラでは通常グラフ彩色に基づいた手法によってレジスタ割り当てを行うことが一般的である．グラフ彩色による手法は高品質な生成コードを得ることが可能であるが，処理に時間を要するという問題がある．そこで，動的コンパイラに向いたレジスタ割り当て手法として，リニアスキャンアロケーションと呼ばれる高速な割り当て方式などが提案されている．

（b）の動的プログラム生成については，現在の汎用マイクロプロセッサアーキテクチャは，動的なコードの生成や変更とあまり親和性が良いとは言えない．たとえば，Java の実行系はプログラムの高速化のために動的

☆1　これに対して従来型のコンパイラを静的コンパイラと呼ぶ．

なコードのパッチ処理などを利用するが，特にマルチプロセッサシステムにおいてこれを安全に行うことは容易ではない．動的プログラム生成をどのように効果的にサポートするかについては今後の課題である．

（c）の動的検査に関しては，コンパイラ最適化によって検査を削除する手法が広く使われている．ただし，すべての検査を最適化によって削除可能なわけではないので，プロセッサ・コンパイラ・実行時システムの連携によって高速な検査を実現することが必要となる．

上記以外にも，ガーベージコレクションの高速化，動的メソッドディスパッチの高速化など，Java を代表とする動的言語の高速化には種々の課題が存在する．

その他の話題

ここでは性能以外の観点から，マイクロプロセッサとコンパイラにかかわる新たな話題について取り上げる．

■ 消費電力制御

現在のマイクロプロセッサはシングルスレッド性能の追求からマルチスレッド性能の追求へ軸足を移しつつある．これは，命令レベル並列性の抽出に限界が見えてきたことが大きな要因となっているが，もう1つの要因としてプロセッサの消費電力が予想以上に増大したことがあげられる．

組み込み向けシステムやモバイル機器では消費電力を削減するためにさまざまな工夫がなされている．コンシューマ向け PC やサーバにおいても消費電力が無視できない課題となりつつある今，コンパイラや OS のサポートによって，より積極的に電力を制御する必要性が生じている．このような方向として，コンパイラ（および OS）制御によって未使用のメモリモジュールを積極的にスタンバイ状態に移行させる技術，与えられた電力指標および性能指標を満たすようコンパイラ制御によって電源電圧および周波数を制御する方式などが研究されている．

■ セキュリティ

もう1つの重要な課題としてコンピュータシステムのセキュリティがある．インターネットを介した攻撃の広まりにより，セキュリティ対策の重要性は増している．

セキュリティ攻撃にはさまざまな手法が用いられるが，頻繁に利用される手法の1つにバッファオーバーフロー攻撃がある．これを防御する手法として，コンパイラにより防御コードを生成する手法が提案され実際のシステムにも適用されているが，その防御は完全ではない．

より広範囲の防御を行うための革新的な手法として，メモリ保護をページよりも小さな粒度で行う方式や，命令セットやアドレスをランダム化する手法が存在する．こういった手法の実現には，コンパイラや実行時システムとプロセッサの密な協力が不可欠であり，低オーバーヘッドで高いセキュリティを実現する手法の開発が望まれる．

将来の展望

マイクロプロセッサの登場から 30 余年，マイクロプロセッサの単体性能は Moore の法則と呼ばれる経験則「プロセッサの性能（元々は半導体の集積度）は 18 カ月で倍になる」に沿って，非常に高いペースで向上を続けてきた．Moore の法則がさらに継続するか否かは今後の技術動向次第であるが，半導体プロセスの微細化に伴う消費電力の拡大など今後の成長を阻害する壁が指摘されている．

Moore の法則と同じ考えをコンパイラ最適化の研究に適用したものとして，Microsoft Research の T. Proebsting による Proebsting の法則がある[5]．Proebsting の法則は「コンパイラによる性能向上は 18 年で倍になる」という経験則である．Proebsting はこの結果をふまえ，コンパイラ研究者はプログラム生産性などの研究にシフトすべきであるとしている．

Proebsting はこの法則を最新のハードウェア上でコンパイラ最適化を有効にした場合と無効にした場合で比較して導いている．しかし，ベクトル計算機や RISC プロセッサなど，アーキテクチャとコンパイラの役割を見直すことによって Proebsting の法則を超えた飛躍的な性能向上が得られることは，これまでの歴史が示している．

また，性能に関する問題だけでなく，先に述べた新しいプログラミング言語やセキュリティの問題など，マイクロプロセッサとコンパイラを代表とするシステムプログラムがより関係を深めることで将来に待ち受ける壁を壊すことができるかどうか，今後の研究の進展が期待される．

参考文献

1）TOP500 Supercomputer Sites, http://www.top500.org/(2005).
2）Muchnick, S. S. : Advanced Compiler Design & Implementation, Morgan Kaufmann (1997).
3）Wolfe, M. : High Performance Compilers for Parallel Computing, Addison-Wesley (1996).
4）Smith, J. E. and Nair, R. : Virtual Machines - Versatile Platforms for Systems and Processes, Elsevier (2005).
5）Proebsting, T. : Proebsting's Law: Compiler Advances Double Computing Power Every 18 Years, http://research.microsoft.com/~toddpro/papers/law.htm

（平成 17 年 9 月 3 日受付）