投機に投資しよう

中島　浩
豊橋技術科学大学

計算機の性能を劇的に向上するための有望な手法として，最近注目されている投機的アーキテクチャを取り上げ，肯定的な立場から議論する．まず投機という手法がこれまでにも成功裏に用いられてきたことと，並列処理では特に有効であることを紹介する．続いて将来の投機的アーキテクチャはハイリスク・ハイリターンとなることが不可避であり，そのためにはさまざまな時間的コストを空間的コストに転嫁しなければならないと主張する．最後に，この時間／空間コスト転嫁は近い将来予想される「トランジスタ・バブル」（大量のトランジスタの使い道に困る現象）の時代にマッチしており，投機こそが最適の投資対象であるとの結論を導く．

アーキテクトは相場師

最近アーキテクチャの世界では，投機というものがはやっている．といってもこれは，研究者や開発者たちが為替や株やオプション先物の売買に血道をあげているという話ではなく，あくまで計算機アーキテクチャについての話である．つまり「何か」（来月の円相場）を予測し，その予測が当たっていることを頼りにうまく立ち回って（円を買って），計算性能を上げよう（大儲けをしよう）という話であり，カッコの中のものとの類似性から投機と呼ばれている．

この投機の対象として最もよく知られているのが条件分岐であり，分岐する（またはしない）ことを予測し，分岐先（後続）の命令のフェッチや実行を分岐命令と並行あるいは先行して行って，制御依存に基づく計算のクリティカルパスを短縮しようというものである．さらに最近は，メモリのアドレスやデータ，演算の結果など，データフローに関する「何か」にも投機の手を広げようという試みもなされている．

投機は伝統芸

さて，投機というといかにも胡散臭く，また「演算結果の予測」などは無茶な冒険のようにも見えるので，アーキテクチャ屋はやることがなくなったあげくに血迷って変なことをやり始めたのかと思われる向きもあるだろう．しかし投機というものは特別目新しいテクニックではなく，これまでに何度も使われてきた伝統的技術なのである．

たとえばキャッシュメモリは，あるアドレスがアクセスされた際に，近い将来同じ（または近くの）アドレスがアクセスされることを予測し，その値を高速なメモリに置いておくことによってアクセス時間を短縮するという技術であり，これはまさしく投機である．したがって仮想記憶やTLB (Tranlasion Lookaside Buffer) など，参照局所性を利用した技術はすべて投機の一種であるといえる．

また命令パイプラインや命令レベル並列処理は，命令間の依存関係がない（あるいは短時間で解消する）ことが「多い」ことを前提としており，この前提が成立することの「静的な予測」に基づくものだと考えれば，（少し牽強付会ではあるが）やはり投機的な技術である．この「こういうことが多い（だろう）からこうする」というアプローチは非常に適用性が高く，命令セット・アーキテクチャをはじめさまざまな方式や技術の基礎となっている．したがって計算機アーキテクチャの発展の歴史は，成功した投機の歴史であるといっても過言ではない．

並列処理とミンコフスキー宇宙

投機の重要性を示す証拠は，並列処理の分野からも見出すことができる．並列処理では一般に，あるプロセッサが今何をしているかを他のプロセッサが知ることは容易ではない．つまりプロセッサ間で協調して何かを行おうとすると，相手の状況を知るための通信が必要となり，そのための時間的なコストを支払う必要がある．この「離れた相手の現在＝自分の未来」というミンコフスキー宇宙のような環境下では，他人の状況を知るためのコストを払いたくなければ「予測」するしかない．

この考えに基づく投機的並列処理の好例が，分散シミュレーションで用いられるTime-Warpの技法である．この方法では分散管理されている「時刻」について自分と他人が同時刻であると予測し，因果関係に不整合が生じないと仮定して投機的にイベント処理を行う．また分散データベースのように，競合が生じる可能性のある処理を無競合を仮定して進めてしまう方法も，やはり投機的な並列処理である．

この他にも負荷分散や探索処理など数々の例が挙げられ，投機という言葉になじみやすい技術は逐次計算よりも多いぐらいである．一般に並列計算の敵役である「問題の逐次性」を投機によって回避できる可能性が多々あり，逐次に比べて並列計算（機）での投機の重要性ははるかに高い．

ハイリスク・ハイリターン

さて投機につきものなのは，リスクとリターンの関係である．失敗しにくい，あるいは失敗しても損が少なく，当たったときの儲けが大きいものに賭けるのが一番ではあるが，そんなうまい話が転がっているかが問題である．これまでの伝統技術としての投機的アーキテクチャは，うまい儲け話を上手に拾ってきた感がある．たとえばキャッシュはよく当たるし，当たれば速いし，外れてもメモリを直接アクセスするのと大差はないし，作るのは簡単だし，まさにローリスク・ハイリターンの典型である．

しかしこのようなバブルはすでにはじけてしまっていて，今後は経済と同じようにハイリターンにはハイリスクが伴うことは必然であろう．この予想は何も計算機アーキテクチャに関する深い洞察から得られたものではなく，「世の中にそんなうまい話が転がっているわけはない」というマーフィの法則的な推論によるものではあるが，よほど我々アーキテクチャ屋が馬鹿でない限り正しいはずである（馬鹿である恐れは十分にあるが）．

したがってリターンの大きなターゲット（たとえば並列計算における問題の逐次性緩和）を選ぶとともに，大きなリスクを見かけの上で小さくするための投資が求められる．すなわち何もしなければ時間コストとして計上されてしまうリスクを，ハードウェア量やプログラム・サイズといった空間コストに転嫁して，ハイリスク・ハイリターンの時代を乗り切ろうというのが，この小文のタイトル「投機に投資しよう」の意味である．

当たるも八卦

このリスクの時間／空間転嫁の方法としてまず考えられるのが，失敗する確率を小さくすること，すなわち予測精度を上げることである．これは真当な方法であって，しかも地道な努力で時間リスクを大きく下げられる場合もしばしばある．分岐予測を例にとると，1%未満という僅かな予測精度の向上によって，性能が10%以上も向上するという報告もある．

ではどうやって精度を上げるかであるが，その前に現状の予測技術がどのようなものかを見てみることにする．現状の予測技術は，プログラムの実行前に予測を行う静的なものと，実行中に予測する動的なものに分けられるが，ここではとりあえずハードウェアによる動的予測について考察する．

最も単純なものはキャッシュや1ビット分岐予測と呼ばれるもので，「さっきはこうだったから，次もこう」という機構であり，「予測」と呼ぶのが恥ずかしくなるような単純さである．分岐予測では少し複雑になった「さっきも，その前もこうだったから」という2ビット予測に，異なる分岐命令の履歴を加味する方法が主流であるが，ごく近い過去の情報だけを用いることには変わりない．

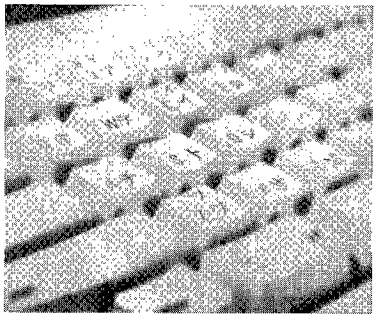
これを為替相場に当てはめると，「昨日（も一昨日も）円が上がったから今日は買い」ということになり，これは「提灯買い」と呼ばれる最低の相場術に相当する．これにせいぜい「マルクが上がったから円を買う」を加えたぐらいで儲かれば誰も苦労はしないわけで，乱高下する最近の円相場ではあっという間に一文なしになるのがおちである．逆に言えばこの程度でそこそこ儲かっている現状は計算機アーキテクチャ進化のごく原始的な段階であって，少し洗練すれば巨億の富を築けるのではないかという期待が持てる．

「投機」顧問とメタレベル命令セット・アーキテクチャ

ではどうすれば，補正予算の規模とか，アメリカの選挙結果とか，中国の洪水とかを考慮に入れつつ，「人の行く裏に道あり」だとか『まだ』は『もう』なり」だとかの洗練された相場術を会得できるのだろうか．端的に言えば，ハードウェアにはそんな難しいことはできないというのが結論であり，それには2つの理由がある．

理由の1つは明らかで，そもそもハードウェアは（巨大かもしれないが）単純な有限状態機械に過ぎず，深遠な論理を実現するのには向いていないという事実である．したがってここはどうしても，賢いプログラマなりソフトウェアなりを顧問に雇う必要がある．この両者のどちらが適任かが問題であるが，一般にプログラマはプログラムが「速くなる」ことに関心はあっても「速くする」ことには怠惰であるので，「速くする」ことに熱心な（人たちが作っている）ソフトウェアであるコンパイラにお願いするのが良策であろう．

もう1つのより重要な理由は，ハードウェアに補正予算の額などが伝わるようにはできていないということである．すなわち，現状の命令セット・アーキテクチャ（ISA）はプログラムの



「オブジェクト・レベル論理」を凝縮したエッセンスとして表現するためのものであって，「この演算は多くの場合こういう結果になる」などといった「メタレベル論理」やヒューリスティクスを表現するようにはできていない．したがって円相場（とマルク相場）の動向だけを追いかけるしか手はなく，補正予算がどうなったかなどは知るよしもない．またどんなに賢い顧問を雇っても，知恵を授けてもらったり情報を提供したりするためのメタレベル命令がなければ，宝の持ち腐れである．

したがってメタレベルのISAを導入することが不可欠であり，これにはいくつかの方法が考えられる．1つは予測に関するさまざまなハードウェア機構を抽象化し，ソフトウェアによる情報収集や指令のために実装から独立した枠組みを構築することである．これは真当で美しいが，多数の研究者が突ついている予測機構を一般的に抽象化できるかどうかは疑わしい．逆のアプローチとしては，現状では不可視であるハードウェア機構をソフトウェアに公開し，低レベルの操作が可能な命令セットとする方法も考えられる．この「マイクロプログラム復権」はWisconsin大学のSmithらのアイデアであるが，アーキテクチャの歴史的教訓である「死者は（しばしば）復活する」を信じれば，案外当たるかもしれない．

この両者の中間的（あるいは日和見的）アプローチは，コプロセッサのような一般的な枠組みを用意して，実装に依存したメタレベル操作をすべて押し込めてしまうというものである．このようにメタレベルとオブジェクト・レベルの計算を切り離すことにより，メタ／オブジェクトの並行計算も自然に導入できる．すなわちメタレベル計算に専念する「投機顧問コプロセッサ」を実現することができるのが利点であり，真剣に追求しようかと思っている．

保険とリスク・ヘッジ

見かけ上のリスクを小さくするもう1つの方法は，投機に失敗したときの時間コストを小さくすることである．円相場とは違って我々の投機では「取り返しのつかない失敗」は許されず，当てが外れたときにもそれなりに正しい計算をしなければならない．この正しい計算への復帰に大きな時間コストがかかるようであると，どうしても保守的になってしまい儲け話を見逃すことになる．また失敗に備えて何らかの「保険」をかける必要があるが，この掛金を時間で払わなければならないとすると，結果的にリターンが小さくなってしまう．

現状の「投機保険機構」であるリオーダ・バッファやシャドウ・レジスタは，一定の空間コストを投じることにより掛金の支払いや保険金の受取が定数時間で済むようにはなっているが，保険期間が10^1～10^2命令のオーダと短い．並列処理での逐次性緩和のような長期先物への投機では，これよりも数桁大きな長期保険が必要であり，メモリを含む大量のマシン状態更新の投機的な実行を許容する必要がある．これについてはキャッシュを用いる方法が，筆者らのものを含めていくつか提案されているが，メモリの多重化などの大胆な投資も考えるべきであろう．

また条件分岐に関しては，本命以外に対抗や穴のパスも並行して実行するリスク・ヘッジの方法もいろいろと提案されている．これは保険よりも大きな投資を要し，また一般的な投機への適用性も明らかではないが，失敗コスト削減効果の大きさはきわめて魅力的である．

おわりに：資金は十分

そろそろ紙数が尽きてきたのでまとめると，投機は（並列）計算機の性能を劇的に向上させる有望な方法であり，予想されるハイリスクはコストの時間／空間転嫁により回避可能であるというのが筆者の主張である．ここで問題となるのは空間コストをカバーするための資金力の有無であるが，現実社会とは違って実は心配御無用である．

筆者が以前行ったラフな調査によれば，命令を実行するための「基本的」論理がマイクロプロセッサ・チップに占める割合は年々低下している．これは単に演算ユニットやプロセッサ数を増やすだけではトランジスタを消費しきれないこと，あるいは消費しても性能に貢献できないことを意味している．したがって将来は大量のトランジスタが「余ってくる」ことは明らかで，貸し渋りなどとは無縁の世界が待っているのである．

さてお客様，余剰資金のご活用に最適な商品がございますが，いかがでしょう，ここは1つ投機に投資なさっては．

(1998.12.9)

自己責任の時代

田浦健次朗
東京大学

中島氏の論旨をバッサリ要約すると，

- これまで投機は成功してきた．
- CPUの世界においては余剰資金がある．

⇓

- もっと大規模な投機をしよう．

私見では，その発想の出所は，「勝ち馬にのろう」である．勝ち馬とは何かというと，徹底した逐次意味論至上主義，あるいはそれをさらに押し進めた，バイナリ互換至上主義である．それらが勝ち馬であったことはVLIW (Very Long Instruction Word) の衰退，スーパスカラの台頭，過去の資産の保存

に徹底したIntelアーキテクチャの躍進，などに見てとれる．これはちょうどコンピュータ業界における「護送船団方式」とでも呼べる方式であった．つまり，これまでの顧客の資産（C/Fortranソース＋バイナリ）の安全を保証しつつ，漸増的に成長していこうという政策である．

日本経済はこの護送船団方式によって衰退した．そして今は金融ビッグバンを起こし，自由な市場と自己責任という発想に，ようやく変わりつつある．このままだとコンピュータ業界においても同様のことが起こる．並列性という現実を無視して，護送船団に守られて自己責任（明示的な並列化）を怠ったソフトウェア業界が衰退する．こんなことを言うと，「護送船団方式は国民（プログラマ）のためだ．国民は昔から自己責任をとろうとした試しはない．国民が馬鹿だから悪いのだ」という大蔵官僚（アーキテクト）からの反論が聞こえてくる．しかしそれは国民のせいではなく，自己責任をとるための自由な市場（明示的な並列化）を行うためのSMP (Symmetric MultiProcessor) プラットフォームが不当に高価であり（4ウェイSMP PCはPC×4よりもずっと高い），その結果国民の購買意欲をそいできた結果である．実際には4ウェイSMP程度の並列性を生かせる応用は利用者が身近に使うものに限定してもいくらもある．今後，公共投資という名の下に税金（資源）の無駄遣いが行われることはなんとしても避けねばならない．必要なのは自由な市場と小さな政府（セーフ）である．護送船団方式が本当に破綻する前にビッグバンを起こし，国民は自己責任の概念を学び，政府が正しい税金の使い方をしているか，監視しよう．

護送船団方式の限界

主要なCPUは命令セットとその意味を変えずに5年間で10倍高速化した．それらはしばしば，アウトオブオーダ実行や投機実行などの洗練された命令レベル並列化技術（公共投資）の賜であるかのような印象を世間に与えているが，実際にはクロックアップ（中小企業の頑張り）が大部分を占めていて，命令の並列実行がこれだけの向上を遂げているわけではない．言い換えれば，同じだけの資源をもっと安直に使えば（シングルチップSMP），それ以上の性能は楽に稼げる．それがなされていないのはひとえにソフトウェアが，単純にCPUが並べられただけの機械を有効活用できていないからだ．

これまでの逐次意味論至上主義では既存のソフトウェアを保護することに重点が置かれていた．アーキテクトが投機を推奨するというのは，アーキテクトに対してであって，そのときソフト開発者は眼中にない．あくまで既存プログラムを護送船団方式で保護するために，自分たちが投機をすることを正当化しているのに過ぎない．その裏の心理は，「俺たちにまかせろ」という発想であり，CPUを単に横に並べるなんていう「芸のない」ことはしたくないという，芸人魂である．

ここで逐次意味論至上主義における「逐次意味論」が何を意味するのかをもう少し明確にして，なぜ護送船団方式がうまくいかないのかを指摘しておきたい．それは同じメモリ上の場所に対する操作をread/writeという粒度において保存する，ということを意味する．つまり，「あるプログラムの並列実行においてメモリ上の各場所に対して行われるread/writeの列が，各writeと，各メモリアクセス（read or write）の順番を逐次実行と同様に保つ」ならば，その並列実行は逐次意味論を保存するというものである．実際にはこの条件は並列実行が逐次意味論を保存するための十分条件ではあるが，必ずしも必要ではない．それどころか実際に手で逐次コードを並列化していて，この条件を厳密に満たすようにプログラムを並列化することはほとんどない（例外: 単純な配列計算）．実際のスケーラブルな並列処理ではもっと高次の知識を用いて，並列化可能なための条件を緩和する．具体的には，複数の操作があるひとかたまりの処理の原始性を仮定すれば入れ替え可能であることを利用したり（典型例: カウンタに対する複数の足し算は，fetch-and-addを原始的に実行すれば，入れ替え可能），同じ領域に対する2回のwriteを1回にマージしたり（典型例: 描画アプリケーションで複数のスレッドからの再描画要求をマージして，一度だけ再描画する），さらにもっと高次の知識を利用したりする．これらの並列実行は，CPUからは逐次意味論を保存していないように映るが，人間には許容できるのである．

投機を大規模化していくと，取り出す並列性の粒度は必然的に大きくなる．すると必ずそれらの処理の間には，上にあげたような，「人間によってはOKだがCPUにはそうとは理解不能な」メモリアクセスが頻発してしまう．大規模な並列実行をしようと思えば，複数の処理の「交換可能性」の概念を，最終的にはプログラマの助けを借りて変更するしかない．その変更の仕方はプログラムによって大いに異なり，read/writeのレベルでそれを保存しようとするやり方はあまりに厳密すぎて成功確率が低く，しかもそれは人間がプログラムの並列化を達成するためには必要ではない．

ビッグバン

明示的な並列化のもたらす利益＝スケーラブルな性能向上をプログラマに伝え，ソフトウェアを書く際の意識改革を促そう．まず，各CPUベンダには，1つの命令セットで，かたや資源を投機実行のために最大限に費やしたCPU，かたや単純な構成でそれらを横並べにしたシングルチップSMP（資源の量は同じ）を両方売ってもらおう．これまでの逐次バイナリは前者の方が速く動くかもしれないし，SPEC intもそうだろうが，明示的に並列化されたアプリや，複数アプリの並列実行には後者の方に部がある．利用者はどちらを選ぶだろうか？　前者を買うことの利点は一部の逐次アプリは速く動くかもしれないということ．しかしそれらはたぶん，MS-WordやNetscape（特にJava）の起動時間を短くするのには大した貢献はしないだろう．かた

や後者の方は普段はおとなしめだが，並列makeやPhotoshop，MP3のエンコーディング，裏でmakeをしながらMS-Wordで原稿を書くなんていうことに貢献する．前者は，さらに速くするにはベンダの新バージョンを待つしかないが，後者はCPUモジュールをさらに買い足せばよい．前者だってCPUモジュール買い足せばいいんじゃないの？　と思うかもしれないが，それはソフトがそうなっていたらの話である．護送船団方式ではそのようなソフトは育たない．

もちろん今の並列プログラムの開発環境でそのまま「自己責任を！」といったところでアプリベンダが重い腰を上げるわけではない．SMP対応アプリを作ろうとしても，現在のプログラミング言語，環境では並列化のコストも高く，ソースの維持も困難になる．そしてさらに悪いことに，そうやって一生懸命作ったところで，ユーザがSMP機を持っていなければ，そんなアプリは多くの人に重宝される機会がない．アプリがないからSMP機も売れないというデッドロックに陥っている．

明らかに並列プログラミング環境の充実，それも逐次の資産に対して一定の尊重を見せる並列プログラミング環境や言語が必要である．それは端的に言えば，C/C++言語を手っ取り早く手動で並列化できるものである．いわゆる自動並列化のような「自動で並列化できるならばするけど，そうでなければ何もしない」などという自己顕示欲まるだしの道具ではなく，一般のC/C++の任意の文を明示的に並列化でき，かつネストした並列性の動的な生成を許す，控えめだが役に立つ道具が必要なのである．これから筆者は，動的な細粒度スレッド生成を万人が使えるようにするためのC/C++用ライブラリを世の中に問うつもりである．次の短期的な目標としては，その並列化が「プログラマが何らかの方法で宣言したある意味において」正しい，つまり逐次の意味を保存していることを検査する方式を開発したい．これは相変わらず最終的な責任をプログラマに負わせるという意味で，「明示的な並列化」であるが，現在行われているただの並列化と異なり，複数の並列に行われた操作間の交換可能性や，結合の方法（複数の描画リクエストは1つを除いて無視するなど）を明示的に表現する方法，そしてその条件の下で並列化の正しさを静的，あるいは動的に検査する道具を備えている．このような道具がそろえば，アプリベンダやアマチュアプログラマも気軽に並列化を行い，結果として，凝ったCPUを2ウェイにしたPCと単純なCPUを8ウェイにしたPCならば，後者が売れる．

小さなセーフ

以上いろいろ考えてみて分かったことは，投機は実はアーキテクトの趣向である．並列化が正しいための条件を唯一に規定するのみならず，違反が起きたときにはせっかくやった計算を，プログラマの見えないところで取り消すなんていうことまでやろうとしている．これまさに「大きな政府」の考え方である．そうではなくCPUは，明示的な並列化をさせ，それが正しいかどうかを検査しながら実行する，最低限の安全性（セーフティ）を保証する機構だけを備えるべきである（アーキテクトはこのくらいじゃ満足できませんかね?）．つまり「小さなセーフ」の考え方である．並列化の正しさや，正しくなかったときの処理はソフトウェアに処理させる，というのがこの考え方である．

こうしてみると，現状の最も大きな問題点は，アプリベンダ，プログラマ，言語屋を含めあらゆるソフトウェア開発者に対して，選択の余地が与えられていないことだと気づく．各ベンダはCPUをいくつも出さないから，ソフトウェア開発者にどの方向性のCPUを選ぶかという権利はない．アーキテクトが「さあ，僕の作ったCPUで君は何を作る?」と得意気になって渡すCPU，それもツボにはまらないと速度の出ないCPUを，彼らの主張する「本来の速度」に近づけるために奴隷のようになって働くだけである．したがって裏で多額の資源が無駄に使われていることを知るよしもない．アーキテクトはなるべく多くのことを，ソフトの気づかないところでやって満足しようとする．そして速度が出ないとなると，「エ?　アー，そりゃここはコンパイラが最適化しないとだめでしょ～」といって逃げるのが常套手段である．

これでいいのか，日本の将来?

(1999.1.10)

甘い話に気をつけよう

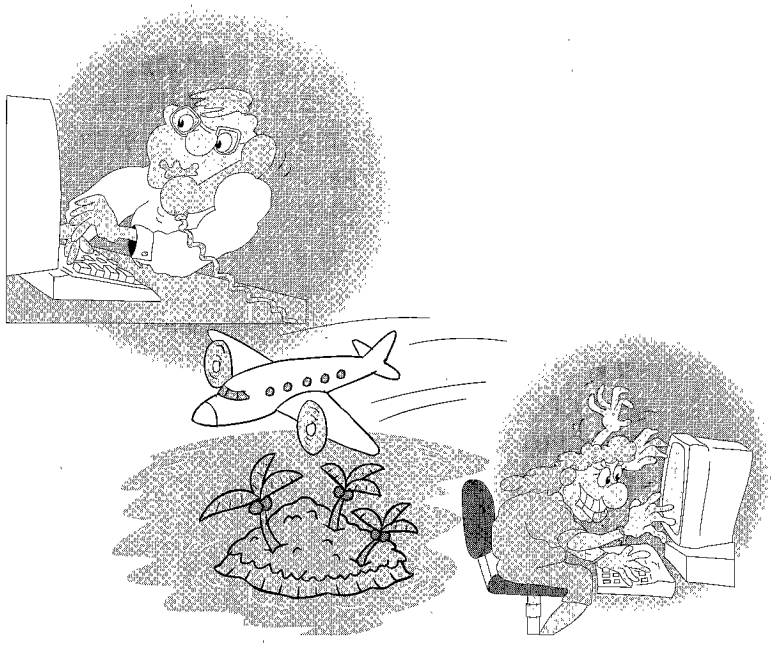
村上 和彰
九州大学

私は投機とかギャンブルとかいった類は「大」が付くくらい嫌いである．というのは真っ赤な嘘であるが（実は，実世界でもベンチャー・ビジネスに手を染めている），ここはディベートを盛り上げる意味で「アンチ投機」派で通そう．

前掲の中島氏の主張は以下の3点に集約されよう．

(1) 計算機アーキテクチャの業界では投機は何も胡散臭いものではなく，すでに伝統芸の域に達しており，計算機の性能向上にきわめて効果的である．

(2) 投機はハイリターン（性能向上）を求めるがゆえにハイリスク（時間コスト）も伴うが，それは空間的なコス



ト（ハードウェア量やプログラム・サイズ）に転嫁可能である．

(3) しかも，その空間コストを支払うだけの資金（チップ上に搭載可能なトランジスタ数）は余剰ともいえるほど十分にあり，したがって投機に投資しない手はない．

上記に対して，私は次の3点から反論を試みたい．

(1) 手持ちの資金（トランジスタ）を投資する側として，投機は本当に良い投資先であるか？　他にもっと良い投資先があるのではないか？

(2) 投機に伴うリスクは小さくない！

(3) 仮に投機に投資するとしても，投資先は厳選すべし！

すなわち，「『投機に投資する』のはそんなに甘い話ではありませんよ」と逆主張したい．

まず，計算機アーキテクトは今までにトランジスタをいったい何に投資してきたのだろうか？　当初は，乗算器や浮動小数点演算器などの機能ユニット，メモリ管理ユニットといった「機能の充実化」に重点投資してきた．そして，ある程度機能が充足してきた段階で，パイプライン制御（昔はパイプライン・プロセッサのことをlook ahead processorつまり先見プロセッサと呼んでいた），分岐予測やキャッシュ・メモリといった古典的な投機技術に投資し始めた．これら古典的な投機技術は，中島氏の表現を借りれば「提灯買い」程度の相場術だが，ローリスクでハイリターンが期待できた．今日の主要投資先は，さしずめスーパスカラ・プロセッサにおけるアウトオブオーダ（out-of-order）実行であろうか（アウトオブオーダ実行自身の歴史はIBM 360/91まで溯り，30年以上の実績があるが）．さらには，トレース・キャッシュやトレース・プロセッサといったハイリスク筋が今後の投資対象として浮上してきそうである．

それでは，これらの投資先は費用対効果の点で果たして優良銘柄たり得ているのだろうか？　たとえば，今日の高性能マイクロプロセッサは4命令同時発行可能（4ウェイ）なスーパスカラ・プロセッサが主流だが，そのリオーダ・バッファ（アウトオブオーダ実行で投機的に実行した命令の実行結果をイン・オーダでレジスタやメモリに反映させるための，中島氏の言うところの「投機保険機構」）に実に約100万個ものトランジスタを費やしている．100万トランジスタ！　ちょっと昔のプロセッサが丸々できてしまうだけのトランジスタ数である．それで，性能は果たしてスーパスカラやアウトオブオーダ実行をやらなかった場合に比べて格段に（あるいは，少なくともトランジスタを投資した分だけ）向上しているのであろうか？　答は否である．4ウェイ・アウトオブオーダ実行スーパスカラで性能向上率はせいぜい2倍程度である．これでは投資額に比べて割に合わない．それだけのトランジスタ／面積を他のもの，たとえばオンチップDRAMとか専用回路，プロセッサ自体の追加とかに投資することも考えてみてはどうだろう．

2番目に，中島氏の「投機に伴うリスクは空間コストに転嫁可能であり，それはチップ上の豊富なトランジスタにより十分カバーできる」という主張に対して，「リスクはそれほど小さくない」と反論したい．氏はもう1つの重要なコスト要因を見逃している．それは消費電力である．リスクが回路という空間コストに転嫁され，その回路が常時動作するとなればその回路が消費する電力は無視できなくなるほど大きくなる．たとえば，MPEG2のデコード（伸長処理）だが，これは一昔前のマイクロプロセッサではソフトウェア・デコードは無理だったのが，今日のアウトオブオーダ実行を駆使した最高性能のスーパスカラ・プロセッサを使えば可能となってきた．しかし，そのためには40W強もの電力が必要だ．一方，専用のMPEG2デコーダ回路を用いればわずか1W弱で同じ処理が可能となる．すなわち，高度な投機技術など一切採用していない低消費電力型の組み込みプロセッサとMPEG2デコーダ専用回路の組合せの方が，高度な投機技術であるアウトオブオーダ実行を駆使したスーパスカラ・プロセッサよりも同一性能という条件下では空間および消費電力コストの面で勝っていることになる．このように投機に伴うコストは決して低くはないのである．

最後に，（中島氏の主張に百歩譲って）仮に投機に投資するとして，一体どんな投機技術に投資したらよいのか考えてみよう．氏は「投機は何でもよし」，特に「投資顧問コプロセッサを推奨する」と主張されている．私は，この「投資顧問コプロセッサ」や先の

「投機保険機構」といった（主に逐次計算機における）投機だけを目的としたハードウェアには投資の魅力を感じない．中島氏は「逐次に比べて並列計算（機）での投機の重要性ははるかに高い」と述べられているので実は認識されていると思うが，「投機実行機構としてのオンチップ・マルチプロセッサ」にこそ投資すべきであると私は考える．投資例としては，我々のリファレンスPPRAM（プロセッサのマルチ化以外に投資したのは全プロセッサから共有されるグローバル・レジスタ・ファイルのみ）がある．類似例としては，（少々過剰投資の感もあるが）Wisconsin大学のマルチスカラ，NECのMUSCAT，などがある．要は，普通の並列処理やマルチプログラミングが実行可能なマルチプロセッサをベースに，できるだけ少ないハードウェア・コストでいかにして「逐次計算の投機実行」を可能にするかである．さらに，並列処理は低消費電力化にも効果があることを付け加えておく．たとえば，電源電圧を1/2にすると単体プロセッサの性能も1/2になるが消費電力は1/4になる．そこで，プロセッサ数を2倍（面積コスト2倍）にすれば，消費電力を1/4×2=1/2に削減しながらシステム性能を一定（1/2×2=1）に保つことができる．

さて中島様，貴方様の大切な資金のご活用に最適な商品がございますが，いかがでしょう，ここは1つ「DRAM混載型のオンチップ・マルチプロセッサ」に投資なさっては．

(1999.1.11)

並列屋こそ投機に投資を

中島　浩
豊橋技術科学大学

後ろから弾が

お二人からの「反応」を楽しく読ませてもらったが，「後ろから弾が飛んできた」ことには少なからず驚いた．というのも，私はアーキテクチャ屋であると同時に（あるいは以前に）並列屋のはしくれであると思っており，同じ並列屋さんから「投機なんかはダメだから並列に投資しろ」と言われるとは思ってもみなかったのである．

最初の拙文にも書いたようにミンコフスキー並列計算空間では，逐次計算に比べて投機の重要性ははるかに高い．この事実をお二人がご存知ないということはあり得ないので，意見の違いは投機への投資方法にあると考える．すなわち私がハードウェアによる直接投資を主張したのに対し，村上氏は「隣のプロセッサで投機せよ」と，また田浦氏は「ソフトウェアで投機せよ」と並列への投資を経由した間接投資を主張されるのであろう．仮にそうだとして，これらのご意見は至極もっともに聞こえ，それでうまくいくのであれば我々並列屋は最高に幸せである．

並列バブルは終わっている

しかしこの「まず並列に投資して」という考え方，極端に言えば「たくさん並べりゃ何とかなるから」というバブル的な考え方はすでに破綻していると主張したい．つまり簡単な逐次計算（機）の集合によって高度な並列計算（機）を実現しようというのがそもそも無理なのである．

たしかにソフトウェアによっても投機的並列処理は実現可能であり，すでに紹介したようにさまざまな手法が知られている．しかし投機の重要な要素である保険を例にすると，ソフトウェアによる計算状態の保存や復元は時間コストがきわめて高くつき，保守的になって儲け話を逃してしまう結果となることは，これもすでに議論した通りである．したがって何らかのハードウェア投資を伴うような投機的並列アーキテクチャを構築することが，並列処理の安定成長に不可欠であると考えるのである．

なおここでいうハードウェア投資はあくまで「投機インフラ」への投資であり，何でもかんでもハードウェアで勝手に投機してというような，ハードウェア独善的護送船団方式ではないことをご理解願いたい．すでに議論したように予測や投機の是非判断というような難しいことはハードウェアの領分ではなく，プログラマやコンパイラの英知に基づく果敢な投機を，陰でしっかり支えようというのが本意である．

さて村上様，田浦様，並列業界にお勤めと伺いましたが，それでしたらぜひここは1つインフラとして投機に投資なさっては．

(1999.1.20)

～　議論の続きは，次のURLをご覧ください．http://www.ipsj.or.jp/magazine/interessay.html　～