CP-PACS とコンパイラの思い出

中田　育男
筑波大学電子・情報工学系（当時）

CP-PACS プロジェクトが岩崎さんたちを中心として計画され，そのハードウェアを中澤さんが担当されるという話が進んでいた頃，私は中澤さんに「私にもソフトウェア担当として参加させてください」とお願いして，参加することになりました．私としては，日立から大学に移って以来ずっとしていなかったもの作りのプロジェクトに久しぶりに参加してみたいと思ったのです．

世界一の性能のものを作りたい，という計算物理系の利用者，情報系の開発者らの熱気に包まれた検討会は，コンパイラとそれに関連したソフトウェアのことしか知らない私にはよく分からないことも多かったのですが，充実感のある楽しいものでした．私の役割はソフトウェアシステムのとりまとめでしたが，私自身がそこで技術的に貢献したという記憶はあまりありません．ただ，CP-PACS のノードマシンの一つの特徴である疑似ベクトル機構の活用については多少とも貢献できたかなと思います．

CP-PACS は数千個のノードマシンからなる超並列計算機とし，ノードマシンとしては日立が製造している PA-RISC マシンをベースとすることで検討が進められていました．PA-RISC は他の一般のワークステーション用チップと同様に，キャッシュによってメモリとレジスタ間の転送時間の遅れを解消しようとするものでしたが，計算物理学の典型的な問題ではデータ量が多くてキャッシュがほとんど有効に働かず，目標とする性能が得られないことが問題でした．その解決策としてハードウェアグループから提案されたのが疑似ベクトル機構（スライド・ウィンドウ・レジスタ）でした．

その機構は次のようなものでした．

(1) 浮動小数点レジスタを 32 個から 128 個に増やす
(2) 通常の命令でアクセスできるのはその中の 32 個である（その 32 個だけが窓を通して見えると考える）．新設のプレ・ロード命令とポースト・ストア命令を使えば，すべての浮動小数点レジスタにアクセスできる．

(3) 窓をずらす(スライドさせる)命令がある.
この機構があれば, 今見えている窓を通して計算している間に, 後で使うデータを見えていないレジスタにプレ・ロードする命令を発行しておき, ロードが完了した頃に窓をそこにずらせば, ロードに要する時間を隠蔽することができます.

しかし, 最初にこの機構が提案されたときは, レジスタ群を数個のブロックに分け, そのブロック単位で窓をずらすことができる, というものでした. それを聞いたとき私はコンパイラ屋として直感的に, ずらす幅は自由にすべきであると思い, それを提案して採用してもらいました.

この機構を生かすコンパイラのアルゴリズムは簡単ではないと思われました. 中澤さんに,「コンパイラは大変でしょうね.」といわれたとき, 私はただ「考えましょう.」とだけ言いましたが, 実際にそれを考えてくれたのは, 私と一緒に仕事をしていた山下さんでした. そのアルゴリズムの基本的な部分はソフトウェア・パイプライニングと呼ばれるもので, その当時主としてマイクロプロセッサの分野で研究されていましたが, 我々はそれを知らずに独自に考え出してしまいました. 山下さんは疑似ベクトル機構をもった PA-RISC 用のプログラムを完成させ, いろいろな例で確かめながら, アルゴリズムの改良も行いました. その結果, ループ 1 回ごとにスライドさせる窓の幅はその 1 回の計算に必要なデータの数(ロードの数)とするものになりました. そのプログラムはその後日立が開発した CP-PACS 用コンパイラに組み込まれました.

私は昔から, 最適化コンパイラが実際にどの程度最適化をしているかを簡単に調べる一つの方法として, 行列の掛算のプログラムの最内側ループの目的コードを調べています. そのプログラムは

```
do i = ...
    do j = ...
        do k = ...
            sum = sum + A(i,k)*B(k,j)
```

という形であり, 最内側ループは k のループです. 通常の RISC マシンでは, 最適化を相当頑張れば最内側ループの目的コードの命令数は 8 になりま

す. その内訳は, sum をレジスタに割付けて代入の必要がないようにし, データ(配列要素)のロード命令が2個, 加算と乗算の命令が2個, 次のデータがロードできるようにインデックスレジスタの値を増やす命令が 2 個(最適化を頑張らなければこの命令数が増える), ループ終了判定とブランチ命令が2個, という構成です.

CP-PACS の Fortran コンパイラでは, この命令数が 3.5, それを実行したときのサイクル数が 2.5 になっています. これは私が知っている範囲では最も効率が良いものです. その内訳は次のようになっています. まず k の 4 回分のループ展開をして

```
do k = ...
      sum = sum + A(i,k)*B(k,j)
      sum = sum + A(i,k+1)*B(k+1,j)
      sum = sum + A(i,k+2)*B(k+2,j)
      sum = sum + A(i,k+3)*B(k+3,j)
```

とすることによって, PA-RISC の FMPYADD(floating multiply and add)命令が使えるようにしています(これは乗算と加算を同時に実行する命令であるが, その 2 つの演算は別々のデータに対して行うものであるからもとのループの形のままでは使えない). このループに対するロード命令は8個, 演算命令は FMPYADD が4個, 窓を8だけずらす命令が1個, ループ終了判定とブランチ命令が1個, で合計 14 命令です(PA-RISC では次のデータがロードできるようにインデックスレジスタの値を増やす命令はロード命令の中に組み込むことができるので, それを利用すれば命令数としては必要ない). これを展開数で割った 14/4=3.5 がもとのループの 1 回あたりの命令数です. ここでのロード命令は, ループの繰返しの後のほうで必要とするデータをあらかじめロードしておくプレ・ロード命令です. 演算命令は以前のプレ・ロード命令によって既にレジスタにロードされているデータに対する演算命令で, したがって, プレ・ロード命令と演算命令は並列に実行され, 演算命令の時間はプレ・ロード命令の時間にかくれてしまいます. したがって実行のサイクルはプレ・ロード命令の8, 窓をずらす命令の1, ループ終了判定とブランチ命令の1の合計の 10 で, もとのループ 1 回あたり

のサイクル数は 10/4=2.5 となります.

このプロジェクトで学んだり考えたりしたことや, 勉強不足を感じて勉強したことなどは, 拙著「コンパイラの構成と最適化」に盛り込むことができましたし, その後私が研究代表者として進めた「COINS コンパイラ・インフラストラクチャ」プロジェクトにも役に立ちました. CP-PACS プロジェクトに参加させてもらったことで, 久しぶりの充実感も得られましたし, そこで得たものは, その後の私の研究にも大きな支えになったと感謝しています.