CP-PACS の PVP-SW 方式誕生のいきさつ

中澤　喜三郎
筑波大学電子・情報工学系(当時)

CP-PACS の思い出話について何か書くようにという御依頼ですので，ここには，筆者が関係した極くざっくばらんなエピソードを披露させて頂く．

分散メモリ型超並列機である CP-PACS システムの大きな特徴として，

- node CPU に PVP-SW（pseudo vector processor based on slide windowed registers）方式と称するアーキテクチャを導入し，擬似的に microprocessor であっても vector processor 並みの高性能を実現したこと
- node processor の相互結合ネットワークに cost performance の良い hyper crossbar network を使用したこと

を揚げることができる．

ここでは，CP-PACS の独自の方式であり，本開発を通じて新たに考案された PVP-SW 方式が誕生した経緯を書き留めておくこととしたい．

CP-PACS プロジェクトのスタート時に，計算機工学分野のメンバーも参画することになったが，その際に，筆者らが意識していたのは，ある応用のための専用機ではなく，一般の科学技術計算にも充分能力を発揮するものを狙いたい，そしてその時点で充分確立した技術と云えるものではなかった超並列計算機の分野の技術の前進を図りたい，cost performance 的にも充分一般に導入可能なものを狙いたいと言うことであった．

しかし限られた予算面からも，大学がもつ開発力からも，世の中一般製品化されているものと基本的なところで上位互換性を保てるものである必要があり，おのずから一般の computer 技術の発展の方向に合致したものでなければならないことも強く意識していた．

一方，CP-PACS に先行した QCD-PAX での経験から，QCD 計算のグループでは CP-PACS では 相当高性能なマシンの開発が期待されて

いた.

そこで, 確か 1991 年の夏, 予算が付きそうな感触が得られた時点で, 初歩的なシステムの構成と, QCD 計算の核になる部分の性能予測を募ったところ, 最初に某会社から提案のあったものではその性能は期待されていたものとは大きな隔たりのある低いものであった.

幸か不幸か, この不満足な性能がわかった日の夜は, プロジェクトのメンバーによる夏休み直前の”暑気払いのコンパ”をやる日であったが, 会が進むほどに, この性能不足のせいで皆の気分は沈み込み, “暗黒の暑気払い”として記憶に残る日となってしまった.

そこで, これではいけないと, 筆者も夏休み中, 性能が出ない理由を自宅で熱心に考えることになった.

提案されていたシステムでは node CPU は;

- 基本の命令 set architecture は HP 社の HP RISC architecture であるものの,
- node CPU は 2 命令同時 issue の super scalar RISC microprocessor で, clock 周波数もその時点での先端な 150 MHz, であった.
- pipeline 化された演算器も浮動小数点とは別々に固定小数点用も独立に用意されていて,
- Multiply and Add 命令も備わっていて,
- cache memory も通常のワークステーション向けのものに比し大幅に強化を予定されていた.

ということで, 一応, 先端的な汎用の microprocessor であった.

しかし, QCD 計算の核の部分での性能予測の結果は 理論的に期待される peak の 300MFLOPS はおろか, その 20~30 % 位の性能しか出ていないものであった.

そして, 性能が出ない理由を検討したところ, 最大の原因は, QCD 計算が必要とするような大規模なデータに対する計算の場合には, cache miss が頻発して, 主記憶より cache memory への転送に伴う cache miss penalty が主要原因であることに思い至った.

つまり, 命令の発行, 演算器の pipeline 動作は 充分休み無く peak 性

能が出るように用意してあったとしても，演算器に供給してやるべきデータが間に合わなければ充分な性能が出る筈がないわけである．

実は，筆者はこれと似たような困難を嘗て経験したことがあった．日立，NEC，UNIVAC などで，汎用の main frame computer の付加機構として，内臓型の vector 処理機構(integrated array processor IAP と云う)，micro program を工夫して vector 処理を恰も vector processor のように処理しようという機構のことで，演算器 pipeline を整備して遊ばせないようにして cost の高い vector register を備えることなしに，経済的に vector 処理性能を稼ごうという機構を導入したのである．しかし，この試みは結局あまり成功しなかった．

一方，筆者らも手がけたことがある vector supercomputer の場合を考えてみると，vector supercomputer では cache memory は存在せず，それに代わるものとしてかなり大容量の vector register(VR) が用意されていて，主記憶と VR の間には load/store のための pipeline が複数用意され，演算器が必要とするデータは予め VR に用意しておいて，データは主記憶とは比較にならないほど高速の VR から遅滞無く途切れることなく演算器へ供給されるようにすることで，演算器の pipeline 能力をフルに利用できるようにしている．

つまり，通常の supercomputer では，cache memory より高速な register を充分な個数用意し，しかもこの register と主記憶の間に強力な pipeline を用意して，比較的低速な主記憶 access latency に因る penalty を避けるために，register を指定しての先行 load(pre-fetch)，と，置いてきぼり store(post-store) の命令機能が用意され，主記憶の遅さが表面に現れないようにしているのである． 言い換えれば演算器へのデータ供給能力に大きな差があるところに scalar processor と vector processor では決定的な差があることに思い至った．

問題が発生する原因がほぼ明らかになったので，これを解決するには scalar processor といえども，主記憶との間に何らかの pre-fetch，post-store 手段を設けるべきで，そのためには主記憶との間でデータのやり取りをするための pre-fetch data buffer register をある程度の本数用

意しなければならないという考えに至った.

しかし, floating point register など program から直接指定できる通常の register の個数を単純に増やすことは, ベースとなる processor の 命令 format , ひいては 命令 set architecture に重大な変更を要することになり, 直ぐには採用できないものであることも明らかであった. 結局, pre-fetch data buffer を導入する必要性は明らかになったものの, 導入方法に関しては成案なしに夏休みを終わってしまった.

夏休み明けに大学に戻って, 筆者の研究室で中村宏講師, 大学院修士 1 年の位守弘充君の 3 人で, 休み中に筆者が考えた上記のような所までの説明をし, まだ解決策に達していない旨の話をし, 議論をした.

そして, 議論を終わって一応解散となり, 筆者が自室に引き上げて帰った直後に, 中村講師が筆者の自室に来て,

「先生, 位守君が”register の本数を増やせば良いというのなら, SUN MICRO の SPARC のような register window なんて云う方法も考えられるのでは“ と云っていますが, これでは駄目ですよね. 」と告げたのです.

それを聞いて, 筆者も 「そうだな. 」と最初は生返事をしていたが, 中村講師が部屋を出た直後に, ”待てよ. これは上手くいくかも知れない ! “と急に思いついた.

そこで, 直ぐに floating point register の数を拡張して, register window 的な構造を導入し,

- 別途 pre-load, post-store 命令を設け,
- 主記憶を擬似的な pipeline memory としておく

ことで, base processor とは architecture 的に上位互換性を保って拡張したことになり, 一応の解決策となる PVP-RW (pseudo vector processor based on register window)方式の仕様書を書き上げた.

ここでは, register window の構造(global area, overlap area, local area などの register の個数)は殆ど固定的なものであった. それでも, application program の中の Do-loop 部分を loop unrolling で modulo scheduling 手法で擬似 vector 化して扱えば, vector processor 並みの性能が発揮できる方式であった.

この時点ではまだ，この機能を活かして性能を稼ぐには，最悪の場合 application program はユーザの hand coding に頼るのも止むを得まいと思っていた．

しかし，ユーザが機械語のレベルでプログラムを hand coding しなければならないのでは，如何にもお粗末であるので，早速，compiler の権威である中田育男教授に PVP-RW の案を纏めたものを見せて，その妥当性と compiler support の可能性について相談をした．

中田教授は，この方式には大方において賛意を表してくれたが，PVP-RW では register window の window size も，offset の採り方もハードウェアで固定的であって，あまり自由度の無いものであったのだが，これらを program で任意に指定できるようにして一般性を持たせるようにすれば compiler support も可能になると思われるので，そうしたらどうかという更なる改良案の示唆をいただいた．

そうして纏め上げたのが，任意の場所へ window をスライドして持っていくことが出来る PVP-SW（pseudo vector processor based on slide windowed register）方式である．

研究室の何人かの学生さんには，この PVP-SW 方式で modulo scheduling 方式がうまく行くかどうかを検討してもらい，一方，中田研究室，山下研究室の方々には compiler support についてよく検討していただいた．

結局，このようにして誕生してきた PVP-SW 方式は，丁度新たに一般向けの MPU を設計しようという日立製作所の開発フェーズと時期が合致し，その MPU の中に実現され，software のサポートも日立によって行われ，CP-PACS の高性能実現の強力な武器になったのである．

PVP-SW 方式は，後になって考えてみると，結局のところ，その後現れた高度 pipeline 制御を実現した MPU での hardware で有力な技術となった register renaming 技術を複雑な hardware の制御機構を設ける必要なしに software で assist した非常にスマートな方式であったわけで，その後に現れた IBM の Power architecture や，Itanium architecture にも，類似の機構が導入されていることをみると，“暗黒の暑気払い”を契機として

誕生したとはいえ，一般的な computer architecture の発展の方向に沿った方式を編み出し得たものと考えている.