CP-PACS プロジェクトの思い出

青木　慎也
筑波大学物理学系（当時）
筑波大学数理物質科学研究科（現在）

私は1991年8月半ばに筑波大学の物理学系に素粒子理論の助手として赴任した．その時点で CP-PACS プロジェクトは（もしかしたら非公式の検討段階だったかもしれないが）既に立ち上がっていたようであり，（正確な日付は覚えていないが）私もすぐにこのプロジェクトのメンバーに加わった．未だに計算機に関しては素人同然であるが，当時はコンピュータのハードのことはほとんど知らず，CPU という言葉は聞いたことがあったが，レジスタ，キャッシュなどはもちろん，メモリやディスクの区別なども出来ていなかった．そのような素人が超並列計算機の製作という（今ではどこでもやっているが）当時の最先端の研究プロジェクトに参加したのであるから，いろいろ分からないことが多く苦労した．この寄稿文では，CP-PACS プロジェクトにおいて，計算機の素人である私がどのようにプロジェクト内で居場所を見つけ，何らかの貢献が出来たのかを振り返っていきたい．

私がプロジェクトに参加する前に，岩崎さん，中澤さんらを中心に既にいろいろな検討がなされていたようであり，特に，CP-PACS の CPU として PA-RISC を使うことが決まっていた．この CPU は当時の量産 CPU の主流であった単精度の演算器ではなく倍精度の加乗算演算器を持っており，150MHz で動作し，1つの CPU あたり倍精度で 300MFlops の性能であった．この CPU を 1024（後から 2048 に倍増）個並列に繋げたものが CP-PACS である． PA-RISC を使った場合に，格子 QCD（以下では単に QCD と呼ぶ）の計算がどのくらいのスピードで出来るか，が問題であり検討が行われた．岩崎さんは，「QCD の主要部分の計算性能を検討したところ，ピークの10％程度（この数字はうろ覚えなので間違っているかもしれません）しか出ないと分かり非常にがっかりして，検討会の後のビールがとても苦かった．」と当時のことを述べられている．初めて私が参加した検討会は，プロジェクトの正否を握る CPU の性能が出ないことが認識された直後のも

のであり，この問題の解決策として，スライド･ウインドによる疑似ベクトル化のアイデアが中澤さんから提唱された．前の検討の背景を知らなかったのでこの提案のポイントや重要性は分からなかったが，岩崎さんが「これでこのプロジェクトはなんとかなりますね．」という言葉を印象深く覚えている．スライド･ウインドに関しては他の専門家の正確な説明があると思うので，ここでは素人からの“解釈”を述べる．QCD 計算は，他のアプリケーションに比べて，演算に対するメモリ･アクセスの比率が格段に高い．演算器は早くて安価ものがどんどん出てくるが，それに比べてメモリの性能は向上せず，QCD では実行性能がでない．今までの高価なベクトル計算機では，ベクトル化によりQCDでも性能が出たが，PA-RISCというスカラーCPUはその性能をキャッシュに頼っている．しかし，QCD ではキャッシュが機能しない，というのが前回の検討結果であった．1つの解決策は，イタリアの APE グループが採用したように，多くのレジスタを用意して，そこに前もってデータを送り込んだり，また，計算後のデータをレジスタに置いて再利用したりすることで性能向上を図るものである．しかしながら，PA-RISC は互換性のために使えるレジスタの数は32と決まっており，APE のように128ものレジスタを使うことが出来ない．その解決策がスライド･ウインドである．まず，物理的なレジスタを128用意する．PA-RISC が演算に使える論理的レジスタは32であるが，その32の連続した論理レジスタを128の物理レジスタの中から動的に選ぶのがスライド･ウインドである．これは128のレジスタのどこを論理レジスタの先頭にするかで実現される．計算に必要なデータは計算に使われていないレジスタに前もって送っておく（プレロード），あるいは，計算した結果はレジスタに残しておき後からメモリに戻す（ポストストア）ことで，メモリアクセス時間の遅れを隠蔽するのである．32の論理レジスタのウインドが次々とスライドして演算を行うことから，スライド･ウインドという名前がついた．レジスタに前もって置いたデータを次々処理していくので疑似ベクトル化とも呼ばれる．スライド･ウインド機能を PA-RISC に付加することで，PA-RISC との互換性を損なうことなく，また，比較的安価に性能向上が図れる．

スライド･ウインドによる疑似ベクトル化により QCD の実行性能の向上が

見えてきたが，どの程度早くなるかを詳細に調べる必要があった．通常であれば，コードをコンパイラにかけて性能を計測し，演算順序や配列の並べ方などを試行錯誤してチューニングしていくのであるが，スライド･ウインドという新しい機能が加わったため，コンパイラ自体も新たに作成する必要があった．また，スライド･ウインドの説明からも分かるように，データのやり取りを計画的にスケジュールしないと性能が出ない．そこで， QCD のもっとも主要な計算の1つである Mult と呼ばれる部分をアセンブラでチューニングすることになった．Mult は，大規模疎行列をベクトルに掛ける演算である．フォートランしか使ったことのない私にはアセンブラの敷居は非常に高いように見えたが，当時，筑波大にいた山下さんによるコンパイラのためのアルゴリズムの解説が転機となり，物理としての意味が分かっている Mult 部分のチューニングに参加することにした．山下さんの講義では，いろいろな条件（演算，データのロード／ストア，ウインドの切り替え，などに必要なマシン･サイクル数や演算器のレジスタ数など）を与えた時に最適なコードを生成するためのアルゴリズムや，それを実現するための時間遷移のチャート図などの書き方を教わった．この方法では，試行錯誤ではなく，論理的に最適なコードを生成できるので，私の好みにぴったりであった．いろいろな演算パターンにこの方法を適用し最適化の練習をした．そして，その使い方に慣れたところで，“人間コンパイラ”として Mult の最適化に取りかかった．とはいえ，アセンブラ未経験の私がスクラッチからコードを書くのは不可能なので，日立側の作成したかなり性能の良いアセンブラ･コードを基に，いろいろな条件の変化に対応できるアセンブラ･コードを作成した．山下さんに教わったアルゴリズムでまずレジスタの割当をスケジューリングし，それをアセンブラに直していくのであるが，難しいパズルを解いていくような感じで苦労した反面，非常に楽しかったことを今も覚えている．当時は実機がなかったが，スライド･ウインドやメモリ構成を反映したシミュレータが存在したので，出来上がったアセンブラ･コードをシミュレータに掛けて性能を測定した．コード上は単体で90％近い性能が出るはずだったが，D-RAM をいくつかのバンクに分けたメモリのバンクコンフリクトの影響などで80％前半の実行性能であった．その後，CPU やメモリの仕様の変更の

ため，単体で70％の後半ぐらいまで性能が落ちたが，これは現在のインテルなどの CPU に比べると驚異的な実行性能である．Mult の実機での性能はシミュレータでの予測とほぼ同じであり，並列化による性能低下を入れても実行性能60％の後半を達成できたように記憶している．（ここは美化して覚えている可能性あり．）このように，計算機の素人である私が少しでも CP-PACS プロジェクトに貢献できたことは，楽しい思い出として残っている．

残念ながら，最近の超並列計算機ではコストの関係上，汎用 CPU をそのまま使うので，CP-PACS でやったような CPU の改変はほぼ不可能である．また，CPU の性能向上に比べてメモリの性能は伸びていないので，CP-PACS の開発時に比べて QCD にとってはますますつらい状況である．並列計算機では並列化による性能低下が大きく問題にされるが，QCD の場合は，まず，単体性能の向上が必須である．現在の汎用 CPU での Mult の単体の実行性能は最高でも30％がせいぜいで，通常は20％台，下手すると10％台になる場合もある．逆に，単体性能が悪いので並列化による低下はそれほど顕著ではない．汎用 CPU の演算性能もキャッシュに頼る部分が多く，スライド･ウインドで経験したような精緻なスケジューリングは望むべくもない．（GPGPU は少し違っていて，ベクトル計算機でやっていたチューニングが必要なようである．）当時はいろいろ苦労したが，今から振り返ると，いろいろな意味でなんと幸福な状況であったことか！