Trabalho 2 - Introdução à Computação Científica

Matheus Sebastian Alencar de Carvalho (GRR20220065) e Tiago Mendes Bottamedi (GRR20220068)
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil

I. Introdução

Este relatório tem como objetivo apresentar o processo de realização do trabalho 2 da disciplina de Introdução à Computação Científica. Tal trabalho consiste em realizar o máximo de otimizações no código produzido para o trabalho 1(Ajuste de Curvas Polinomial com Cálculo Intervalar). Ao longo do relatório serão demonstradas as otimizações consideradas, os motivos pelo qual foram ou não implementadas, seus efeitos nas métricas solicitadas e demais considerações julgadas relevantes.

II. Considerações

- O trabalho foi rodado e testado na máquina H12 do DINF, cuja arquitetura está presente no arquivo Arquitetura.txt
- No teste da v2 com o LIKWID para 10^7 pontos, o LIKWID apresentou o seguinte problema:

```
ERROR - [./src/access_client.c:access_client_startDaemon:161] Cannot allocate memory.
Failed to fork access daemon for CPU 3
ERROR - [./src/perfmon.c:perfmon_init:1544] Cannot get access to performance counters
```

  Mesmo após pesquisas e tentativas de consertar esse problema, ele não pôde ser resolvido. Por conta disso, as linhas da v2 que necessitam do LIKWID vão somente até 10 pontos. Como o tempo é medido sem uso do LIKWID, essa linha vai até 10^8 normalmente.

III. Otimizações

Nessa seção, serão apresentadas as otimizações realizadas no código. O foco principal foi otimizar as operações em ponto flutuante e o acesso a memória, visando diminuir ao máximo o tempo de execução do programa, mas mantendo uma boa precisão. Um dos pontos mais focados foi a tentativa de utilizar AVX para realizar contas em paralelo, no entanto, o modo como as operações intervalares são calculadas(utilizando funções) e, possivelmente, a grande quantidade de vetores utilizados simultaneamente, impediram a utilização desse recurso.

A. Estrutura de dados

Uma das principais otimizações feitas foi a troca da estrutura de dados utilizada. Enquanto na v1 era utilizada Array of Structs(AoS), na v2 foi utilizada Struct of Arrays(SoA). Essa otimização permitiu um melhor acesso à memória, uma vez que os dados passaram a ser carregados de forma contínua. Além disso, a matriz A foi transformada em um vetor, com esse mesmo objetivo.

B. Looping Unroll & Jam

Visando uma melhoria com relação aos acessos à memória, o desenrolar de laços foi feito nas seguintes funções:

- calculaResiduo, com unroll em i, o que diminui o carregamento do vetor de coeficientes;
- calculaAeB, com unroll em i, o que diminui o carregamento dos vetores de soma e de termos independentes;

C. Blocking

Visando o uso completo de linhas de cache já carregadas sem pressão adicional em registradores, o uso de blocking foi feito nas seguintes funções:

- calculaResiduo, com blocking em i, o que gera uma reutilização das linhas da cache que têm o vetor de resíduo e o de pontos x;
- calculaAeB, com blocking em i,gera uma reutilização das linhas da cache que têm vetor de pontos x e o de pontos y;

D. Inversão de laço

Um dos fatores mais importantes para a otimização do código está relacionado à inversão do laço de calcular os somatórios, feito na função calculaAeB. Anteriormente essa função tinha como laço interno o tamanho da matriz dos somatório e no laço externo a quantidade de pontos dado. Após a inversão desse laço, a quantidade de operações que são feitas é reduzida, tendo em vista que, agora, para cada ponto da matriz o somatório é calculado parcialmente e sem repetições de pontos, também influenciando diretamente no tempo de execução do programa.

E. Operações

Outra otimização bastante impactante foi nas operações realizadas. Enquanto na v1 as potências eram calculadas utilizando a função pow, na v2 elas passaram a ser calculadas utilizando multiplicação. Além disso, ao invés de calcular cada potência do zero, a potência anterior foi reutilizada para calcular a seguinte, bastando multiplicar novamente pelo valor desejado (com exceção do cálculo dos somatórios dos Mínimos Quadrados, onde os somatórios da última coluna realizam o cálculo da potência para que possam ser calculados em paralelo com os da primeira linha).

Além disso, como as entradas geradas são sempre positivas, é possível reduzir algumas contas feitas nas operações intervalares, em especial nas funções de multiplicação e divisão, nas quais era necessário realizar todas as contas possíveis e

verificar o menor/maior valor obtido. Ademais, as chamadas de funções intervalares foram substituídas por macros ou pelo próprio código, reduzindo ainda mais o tempo de execução do programa.

F. Considerações

Foi cogitado aplicar um looping unroll na função que realiza as eliminações de Gauss, mas, por ser uma matriz muito pequena, não encontramos melhorias significativas que validassem essa mudança.

IV. Resultados

A seção abaixo apresenta os resultados obtidos através de gráficos e a análise desses resultados de forma sucinta, visando justificá-los. Em todos os gráficos, o eixo das abcissas representa o logaritmo natural do número de pontos. Nos gráficos de tempo, o eixo das ordenadas também é representado de forma logarítmica. Além disso, para o cálculo do sistema foram gerados somente os gráficos de tempo e MFLOP/s, conforme solicitado.

A. Tempo

O tempo para a geração do sistema e para cálculo do resíduo diminuíram de forma exponencial, tendo em vista as seguintes mudanças:

- Calculo de potências de forma ”recursiva”, reaproveitando o resultado anterior e sem chamar a função pow para cada potência;
- Troca da estrutura faz com que o acesso à memória seja melhorado;
- Diminuição de operações desnecessárias, considerando a ausência de valores negativos.
- Unroll de alguns laços, causando um menor número de acessos à memória para alguns vetores, sobrecarregando os registradores mas diminuindo o tempo total de execução.

O tempo de cálculo do sistema não teve mudanças significativas.

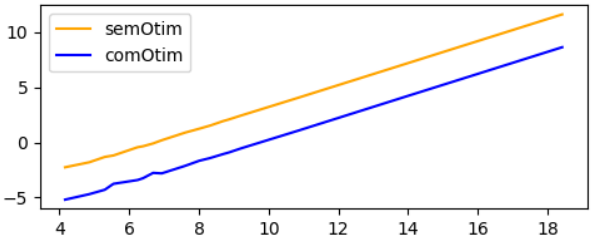

Figura 1. Gráfico do tempo para a geração do sistema

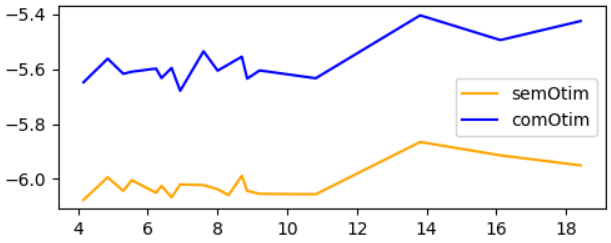

Figura 2. Gráfico do tempo para o cálculo do sistema

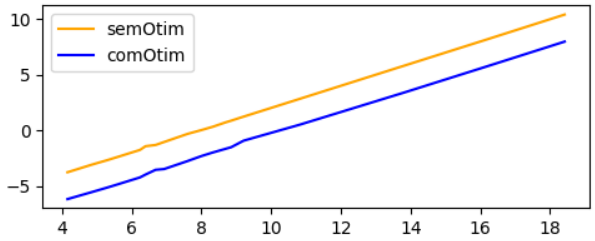

Figura 3. Gráfico do tempo para o cálculo do resíduo

B. MFLOP/s

A quantidade de MFLOP/S, de forma geral, diminui em todos os gráficos, devido a uma maior melhoria na quantidade de operações do que as melhorias de acesso à memoria, ou seja, o programa passa menos tempo realizando contas do que acessando a memória.

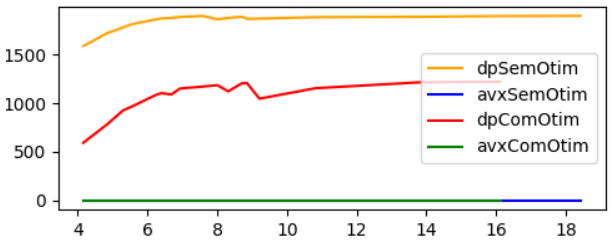

Figura 4. Gráfico dos MFLOP/s para a geração do sistema

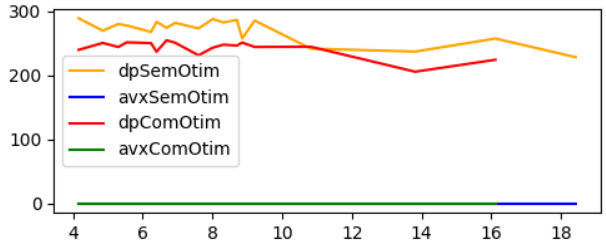

Figura 5. Gráfico dos MFLOP/s para o cálculo do sistema

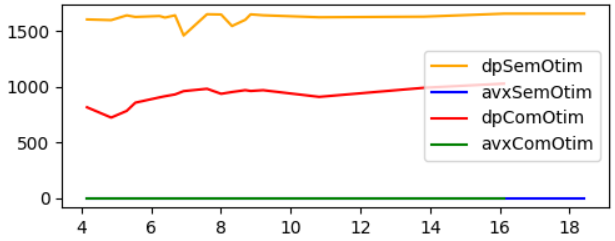

Figura 6. Gráfico dos MFLOP/s para o cálculo do resíduo

C. Cache Miss Ratio

Os valores de Cache Miss apresentam um balanço com a primeira implementação porque a mudança na estrutura de dados faz com que tenha menos cache miss, entretanto, os Loop Unrolls (principalmente o do laço de gerar sistemas, por ser maior) geram o efeito de Register Spill, que consiste na sobrecarga dos registradoers, e de vários vetores tentarem ocupar o mesmo espaço, aumentando o Cache Miss.

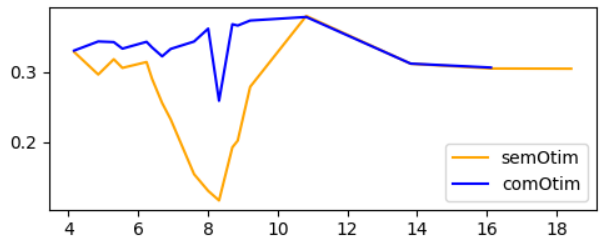

Figura 7. Gráfico do Cache Miss Ratio para a geração do sistema

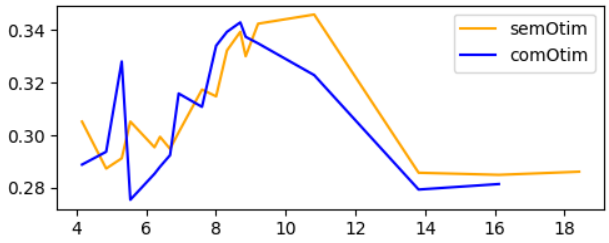

Figura 8. Gráfico do Cache Miss Ratio para o cálculo do resíduo

D. Memory Bandwidth

Os motivos de melhoria de Memory Bandwidth estão diretamente relacionados ao uso de SoA e Loop Unroll. O uso de SoA faz com que elementos próximos sejam carregados juntos (e eles são utilizados juntos), e o Loop Unroll reduz a quantidade de carregamentos.

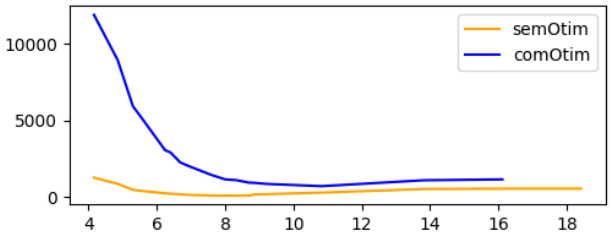

Figura 9. Gráfico da Memory Bandwidth para a geração do sistema

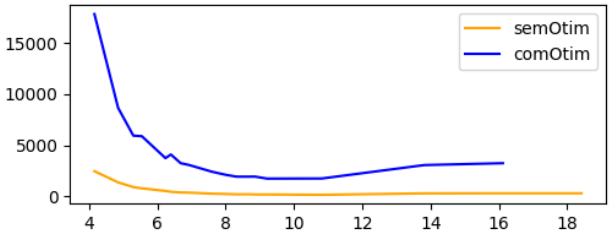

Figura 10. Gráfico da Memory Bandwidth para o cálculo do resíduo

V. Conclusão

Portanto, tendo em vista as mudanças realizadas e as melhorias obtidas, é possível concluir que houve uma otimização significativa no código da v1 para a v2, em especial no que diz respeito às contas em ponto flutuante e ao acesso à memória. Além disso, algumas ”melhorias” testadas causaram na verdade perda de desempenho, principalmente em looping unroll muito complexo, que causava um Register Spill.