Aplicação das ferramentas Intel Parallel Studio para modernização de código para métodos numéricos de diferenças finitas para solução de equações diferenciais parciais em arquitetura Intel Haswel/Broadwell.

Gabriel Pinheiro da Costa[1,2], **Carla Osthoff**[2], **Frederico Cabral**[2]

[1]Centro de Educação Tecnológica Celso Suckow da Fonseca- CEFET-RJ UnED Petrópolis

[2]Laboratório Nacional de Computação Científica - LNCC

Abstract. *This papers aim to show a numerical method code optimization strategy for Intel parallel environments. Will be described the standard and optimized code versions, as well the reasons that led to the development of this new version and the tests that prove the performance improvement and confirm its validity.*

Resumo. *Este artigo tem como objetivo a apresentação de uma estratégia para a otimização do código de um método numérico em ambiente paralelo Intel. Serão descritas as versões padrão e otimizada deste código, as razões que levaram ao desenvolvimento dessa nova versão, bem como os testes que comprovam o ganho de desempenho e confirmam sua validade.*

1. Introdução

Métodos numéricos para solução de equações diferenciais parciais possuem um papel de destaque e especial importância no presente estágio de desenvolvimento tecnológico. Esses métodos levam à solução de equações que modelam problemas de especial interesse na atualidade, em áreas distintas como química, física, egenharias e metereologia.

Felizmente esses e outros códigos são contemplados pela possibilidade do uso de ferramentas de otimização, como o Intel Parallel Studio. Esses softwares filtram informações da série de instruções contidas no código durante a execução, e através delas gera indicadores acerca do comportamento dessas rotinas, indicando possíveis gargalos, excessos de tempo em um determinado conjunto de instruções ou atividades, grau de balanceamento, dentre outros.

Através desses indicativos torna-se possível a elaboração de estratégias e versões otimizadas do método afim de superar os resultados indesejáveis apontados pela ferramenta.

Esse resumo apresentará de forma breve uma pequena síntese de uma estratégia desenvolvida para o método HOPMOC para aumento de desempenho em ambientes paralelos, mais bem detalhada em um trabalho já publicado.

As seções subsequentes mostram a versão padrão do método HOPMOC para solução de equações diferenciais parciais, e uma versão otimizada do mesmo método, impulsionada por indicativos retirados das ferramentas Intel Parallel Studio.

2. HOPMOC Naive

A versão padrão (Naive) do HOPMOC unidimensional opera sobre um vetor de pontos, realizando o preenchimento dessa matriz para cada instante genérico de tempo.

A figura 1 mostra o núcleo da versão Naive à esquerda, composta por 4 loops for, responsável pelo preenchimento da malha de saída em dois instantes genéricos de tempo: t+1/2 e t+1.

A forma mais simples para a paralelização dessa rotina talvez seja através da interface OpenMP, com a inserção de suas diretivas padrões de paralelização, como o *pragma omp for* antes dos loops for.

3. HOPMOC EWS-Synch

```
for (int i = head+1 ; i <= N-2 ; i+=2) {
    U_old[i] = alfa*(U_new[i-1] + U_new[i+1]) + (1 - 2*alfa)*U_new[i];
}
head = (head+1)%2;

for (int i = head+1; i <= N-2 ; i+=2) {
   U_old[i] = (U_new[i] + alfa*U_old[i-1] + alfa*U_old[i+1])/(1+2*alfa);
}
head = (head+1)%2;

for (int i = head+1 ; i <= N-2 ; i+=2) {
   U_new[i] = alfa*(U_old[i-1] + U_old[i+1]) + (1 - 2*alfa)*U_old[i];
}
head = (head+1)%2;

for (int i = head+1; i <= N-2 ; i+=2) {
   U_new[i] = (U_old[i] + alfa*U_new[i-1] + alfa*U_new[i+1])/(1+2*alfa);
}
```

```
lock[nid] = 1;
#pragma simd
    for (int i = inicioLocal+head+1 ; i <= finalLocal; i+=2) {
    //ANNOTATE_ITERATION_TASK(loop_HOP_EXP_1);
        U_old[i] = alfa*(U_new[i-1] + U_new[i+1]) + (1 - 2*alfa)*U_new[i];
    }

lock[nid] = 0;

while ((lock[nid+1] == 1) || (lock[nid-1] == 1));

head = (head+1)%2;

lock[nid] = 1;
#pragma simd
    for (int i = inicioLocal+head+1 ; i <= finalLocal; i+=2) {
    //ANNOTATE_ITERATION_TASK(loop_HOP_IMP_1);
        U_old[i] = (U_new[i] + alfa*U_old[i-1] + alfa*U_old[i+1])/(1+2*alfa);
    }

lock[nid] = 0;

while ((lock[nid+1] == 1) || (lock[nid-1] == 1));

head = (head+1)%2;
```

Figure 1. Núcleo do HOPMOC Naive(esquerda) e núcleo do HOPMOC EWS (direita)

As análises realizadas através das ferramentas Intel mostraram que a paralelização pura através de diretivas OpenMP esbarrava em certas limitações que reduziam significativamente o desempenho do método. O tempo gasto para o escalonamento de threads para os segmentos da matriz, e principalmente o tempo gasto em barreiras de sincronização esperando o fim do processamento de outras threads se mostraram obstáculos consideráveis.

Em resposta a esses gargalos, e com base nos dados coletados, foi desenvolvida outra versão do HOPMOC visando o ganho de desempenho: o EWS-Synch (Explicit Work Sharing - Synced). Essa estratégia consiste em uma versão com distribuição manual(explícita) de threads, o que reduz o tempo gasto para scheduling em tempo de execução. A malha de pontos é dividida de forma explícita em partes de tamanho fixo, sendo cada uma dessas partes designada a uma thread, de forma permanente até o fim da execução.

A versão EWS-adSync minimiza a troca de mensagens entre as threads e o tempo gasto em barreiras de sincronização. Nessa estratégia uma determinada thread precisa somente das informações providas por suas threads vizinhas(esquerda e direita), para a continuidade da execução. A sincronização entre essas threads é feita através de uma matriz unidimensional de valores booleanos, em que cada entrada corresponde a uma thread, e o valor dessa entrada determina se os dados dessa respectiva thread podem ou não ser acessados pelas threads vizinhas naquele instante.

4. Resultados e Discussão

Foram realizados testes em uma máquina contendo uma Intel Xeon CPU E5-2698 v3 @ 2.30Ghz composta por 32 núcleos físcos. A figura 2 mostra um gráfico de speedup por número de cores das versões Naive, CoP(não explicitada nesse artigo) e EWS-Synch.

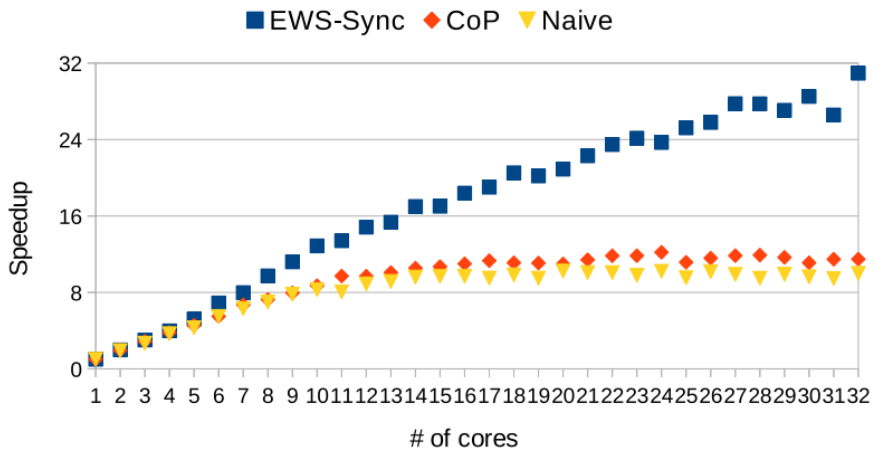

Figure 2. Speedup por utilização de cores

O gráfico mostra o baixo ganho de speedup da versão Naive, que atinge seu valor máximo de 10 com apenas 15 de 32 cores físicos, enquanto a versão EWS-Synch continua a apresentar um ganho de desempenho quase linear, que se estende até a utilização de todos os 32 cores físicos, alcançando um speedup máximo próximo de 32.

O ganho de speedup, bem como de outras melhoras observadas como a de distribuição de carga, comprovam a validez das estratégias desenvolvidas à partir dos levantamentos das ferramentas do Intel Parallel Studio, reafirmando a importância e a praticidade do software e seu papel ativo no desenvolvimento e otimização computacional.

5. Referencias

References

Cabral, F. L., Osthoff, C., Costa, G. P., Brandão, D., Kischinhevsky, M., and de Oliveira, S. L. G. (2017). Tuning up tvd hopmoc method on intel mic xeon phi architectures with intel parallel studio tools. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 19–24. IEEE.

Cabral, F. L., Osthoff, C., Costa, G. P., de Oliveira, S. L. G., Brandão, D., and Kischinhevsky, M. (2018a). An openmp implementation of the tvd–hopmoc method based on a synchronization mechanism using locks between adjacent threads on xeon phi (tm) accelerators. In *International Conference on Computational Science*, pages 701–707. Springer.

Cabral, F. L., Osthoff, C., Souto, R. P., Costa, G. P., de Oliveira, S. L. G., Brandão, D., and Kischinhevsky, M. (2018b). Fine-tuning an openmp-based tvd–hopmoc method using intel® parallel studio xe tools on intel® xeon® architectures. In *Latin American High Performance Computing Conference*, pages 194–209. Springer.