ENGENHARIA AEROESPACIAL E MECÂNICA
MECATRÔNICA
CE-265 PROCESSAMENTO PARALELO

Docente: Jairo Panetta
Discente: Lucas Kriesel Sperotto

Exercício 6 – RELATÓRIO E CRÍTICA

Paralelização por CAF do "Jogo da Vida" no Sistema "CROW" & Sumário e Crítica de Trabalho Científico

Este relatório esta dividido em três partes distintas: o relatório da paralelização, o resumo e crítica do artigo de Hochstein et. al. e os anexos.

O anexo se encontra em arquivo "JogVidaCAF.tgz" anexados na submissão do trabalho, contendo os códigos fonte paralelizados, o Makefile, os Scripts de execução e os arquivos de retorno dos resultados para 4, 8 e 16 "imagens".

Paralelização CAF:

Inicialmente verifiquei o funcionamento do software e como ele tratava as matrizes "coarrays". Visto que cada matriz era criada com três colunas, uma principal e duas "ghost zone." Após terminar a análise, tentei basicamente de quatro formas paralelizar o trabalho do procedimento "UmaVida()", não irei descreve-las, apenas ressalvo que a ordem lógica foi muito interessante e bem menos tediosa que a paralelização por MPI. Também renomeei os arquivos de saída, colocando apenas ".out" para manter o "Notepad++" como aplicativo padrão para a visualização dos mesmos.

A primeira coisa foi colocar uma diretiva "sync all" no final do procedimento uma vida, para manter sincronismo no final do procedimento e garantir que todas as "mensagens" fossem trocadas. O raciocínio que tive por final é que a verificação dos estados deveria ser feito localmente em cada tabuleiro (no coarray da própria imagem), e gravado tanto localmente quanto nas respectivas "ghost zones" das imagens vizinhas (imagem "i" grava sua coluna "útil" na ultima coluna da imagem "i-1" e na primeira coluna da imagem de "i+1"). Para isso, separei com um "if" a execução para processos menores que o ultimo e maiores que o primeiro, dessa forma eles podiam gravas a sua linha nas "ghost zones" dos anteriores e dos próximos sem problemas.

Com isto feito verifiquei o funcionamento e, um segundo teste, foi trocar a diretivas "sync all" por "sync images()" de forma que o sincronismo ocorra entre duas imagens. Comentado em aula que

isto otimizaria o código por parte do compilador, coloquei as diretivas junto com os “if’s” comentados anteriormente. Não tenho certeza, mas parece que obtive um pouco a mais de desempenho, ao menos comparando os tempos de execução retornados pelo “crow” (não medi o tempo diretamente mas deveria ter feito isso).

Para garantir o correto funcionamento da paralelização, gerei casos de teste com a função “DumpTabuleiro()” descomentada de forma a ver toda a evolução do veleiro, para de 4 a 16 “imagens” e verifiquei uma a uma o percurso do veleiro, nos arquivos de saída gerado para a entrega, apenas mantive a “DumpTabuleiro()” do estado inicial e final do tabuleiro.

Síntese e Critica de Trabalho Científico:

Este trabalho se inicia enfatizando que métricas para o levantamento do tempo de soluções em computação de alto desempenho (que leva em conta o esforço humano da codificação e o tempo de maquina) não davam grande importância ao esforço humano necessário à paralelização, principalmente para programadores inexperientes. Medidas de numero de linhas de código eram usadas como referência ao esforço necessário, fato que o autor considera ineficiente frente aos poucos estudos empíricos realizados até o momento.

Dessa forma, foi apresentada uma metodologia para a observação e levantamento dos dados necessários na quantificação da complexidade de tempo e do esforço humano exigidos em uma codificação para computação de alto desempenho. Para isto, foi comparada quatro famílias de problemas em diferentes modelos de programação paralela, levando em conta o grau de familiaridade dos programadores iniciantes (estudantes) com a computação paralela.

Os resultados mostram que programadores novos, são capazes de atingir um bom desempenho em suas aplicações paralelas, enfatizando que codificações em OpenMP não exige muitas linhas de código adicionais , já em MPI é necessário um numero grande de linhas de código a mais que a versão sequencial.

Como numero de linha de código é considerada uma métrica ineficiente pelo autor, um próximo passo dado foi comparar a complexidade da codificação paralela em relação à codificação serial do problema. Nisto foi verificado que o custo por linha de código para o código paralelo (MPI e OpenMP) é maior do que o custo por linha de código para código serial. Concluindo que um maior esforço é necessário para implementar uma aplicação paralela em MPI do que em OpenMP.

Este estudo torna claro que tanto a abordagem quanto o modelo utilizado para a paralelização influi diretamente no esforço humano necessário. Creio eu que pela complexidade tanto para o entendimento da arquitetura (modelo) envolvida quanto a complexidade da própria linguagem (diretivas OpenMP e MPI) utilizada.

Gostei muito do trabalho e da visão do autor, principalmente pelo fato de vivenciarmos isso em sala de aula. Nas execuções das atividades propostas, pude perceber e avaliar as conclusões tiradas pelo autor do artigo. Em cada atividade tivemos desafios para entender o problema, entender o modelo de paralelização proposto e claro a sintaxe da própria linguagem. Realmente digo que codificações MPI exigem um maior esforço se comparado a OpenMP, e agora neste ultimo exercício pude ver que o modelo de programação paralela influi diretamente na complexidade da codificação. Vendo que

CAF tenta minimizar o esforço do programador para com o tratamento da memória (troca de mensagem).

Realmente acredito ser importante considerar o esforço do programador, tanto para viabilizar uma programação paralela, como para estimular o desenvolvimento de linguagens paralelas de “alto nível”, que torne a paralelização mais sutil aos olhos do programador do que é hoje.