XXIII Seminário de Iniciação Científica
XX Jornada de Pesquisa
XVI Jornada de Extensão
V Mostra de Iniciação Científica Júnior
V Seminário de Inovação e Tecnologia

**Modalidade do trabalho**: Relatório técnico-científico
**Evento**: XXIII Seminário de Iniciação Científica

# ANÁLISE DE DESEMPENHO COM A PARALELIZAÇÃO DO CÁLCULO DE NÚMEROS PERFEITOS[1]

**Éder Paulo Pereira[2], Gilberto Przygoda Marmitt[3], Emilio Hoffmann De Oliveira[4], Edson Luiz Padoin[5], Carlos Eduardo Das Chagas Ribas[6].**

[1] Trabalho da disciplina de Processamento Paralelo do curso de Ciência da Computação da Unijuí
[2] Aluno do curso de Ciência da Computação da Unijuí
[3] Aluno do curso de Ciência da Computação da Unijuí
[4] Aluno do curso de Ciência da Computação da Unijuí
[5] Professor Doutor do curso de Ciência da Computação da Unijuí
[6] Aluno do curso de Ciência da Computação da Unijuí

## Introdução

Alguns problemas não são possíveis de serem resolvidos usando a programação sequencial, pois possuem um tempo computacional que é inviável. Neste sentido, surge a programação paralela, onde temos Threads/Pocessos atuando como novos subprocessos de um processo pai, e assim conseguimos 'dividir para conquistar', alcançando resultados mais rápidos e precisos, diferente da programação sequencial. O processamento paralelo tem sido utilizado em várias situações, das quais pode-se citar, a previsão do tempo que interfere em nosso dia a dia, ou ainda, simulações da terra, como o super computador Earth Simulator, o qual foi inaugurado pela NEC em 2002, com a finalidade de fazer simulações de comportamentos da terra, a fim de prever terremotos, dentre outros abalos sísmicos.

Em Matemática, um número perfeito é um número inteiro para o qual a soma de todos os seus divisores positivos próprios (excluindo ele mesmo) é igual ao próprio número. Por exemplo, o número 28 é um número perfeito, pois: 28=1+2+4+7+14.

À medida em que os números vão sendo incrementados, a quantidade de operações de soma e divisão vão crescendo junto, sendo que para chegarmos ao quinto número perfeito (33550336), um computador normal levaria muitas horas, afinal, ele precisaria testar desde o número 1 até o número 33550335, se há divisores com resto zero para o referido número, e armazenar estes números em outro lugar temporário, para depois, realizar a soma de todos eles e então realizar a comparação, para enfim nos dizer se é ou não um número perfeito.

Nosso objetivo, entretanto, foi realizar a verificação de números perfeitos, mas, para isso, estabelecemos que o número limite para os testes seria o 300000, pois, o quarto número perfeito, é o 8128, e nosso computador chega muito rapidamente neste resultado, então não conseguimos visualizar o real ganho do processamento paralelo. Já o quinto número perfeito é muito alto, e como nosso hardware não é tão poderoso, levaria horas, senão dias, para realizarmos os testes, inviabilizando o trabalho, então definimos um meio termo, onde pudéssemos observar as reais vantagens de nossos cálculos.

Para este artigo, utilizamos 3 métodos diferentes para a comparação. Inicialmente, desenvolvemos um algoritmo que realiza os testes de maneira sequencial, testando cada número um a um. Posteriormente, usamos a programação paralela com Pthread, onde paralelizamos o laço de repetição que realiza o teste em si. E por fim, realizamos o último passo usando a API OpenMp, que será discutida posteriormente.

O que podemos notar, é que a programação sequencial para a maioria dos casos, é útil e funcional. Entretanto, para a realização de tarefas mais grandiosas, envolvendo muitos cálculos, não há outro caminho a não ser a programação paralela, o que confirmamos neste trabalho.

Objetivo

O objetivo Deste trabalho, é buscar paralelamente os números perfeitos, mas para nosso trabalho, definimos que o intervalo de busca seria do número 1 até o número 300000. Este número foi escolhido para não ficar muito rápida a execução do algoritmo, mas também para não ficar muito lento, a fim de termos resultados mais concisos. Para reduzir o tempo computacional e conseguir calcular maiores números perfeitos, usaremos diferentes métodos de programação paralela. Primeiramente o problema será desenvolvido com a programação sequencial, e servirá de base para comparar o tempo com os demais. Na sequência será utilizado a programação paralela em memória compartilhada com Pthread E OpenMP.

Números perfeitos

No ano de 325 a 265 aC, existiu um grego matemático chamado Euclides de Alexandria, que, dentre outros tratados, confeccionou o tratado denominado ’Os Elementos’, onde, no livro IX, proposição 36, nos é apresentado um método para encontrar um número perfeito, como segue:

"Caso números, quantos quer que sejam, a partir da unidade, sejam expostos, continuamente, na proporção duplicada, até que o que foi composto todo junto se torne primo, e o todo junto, tendo sido multiplicado pelo último, faça algum, o produzido será perfeito (BICUDO, 2009, P. 349)."

Este método foi descoberto por Euclides, e depois usado como base para a descoberta dos números amigáveis.

Flags de compilação

Quando programamos em C ou C++, conhecemos as flags de compilação, que são instruções que damos ao processador para otimizações diversas. Existem muitas flags, e cada arquitetura de processador usa esta ou aquela flag específica, baseado em seus registradores e outros componentes internos de cada processador. No entanto, há a flag ’O’, que atua como uma flag ’genérica’, funcionando para a maioria dos processadores com um resultado satisfatório. Mas para problemas muito específicos, ou ainda processadores com arquiteturas mais exóticas, existem outras flags que são destinadas à estas arquiteturas de processadores. Para todos os efeitos, no site do compilador Gcc (www.gcc.gnu.org), temos várias informações, cada uma baseada na versão do compilador.

Por último, mas não menos importante, usa-se para programação de alto desempenho, o sistema operacional Linux e suas variantes para cada arquitetura, e em nosso trabalho não foi diferente, como explicaremos no próximo tópico.

Cenário de testes

O nosso cenário de testes possui o seguinte hardware: Um notebook, marca Positivo, equipado com processador Intel Core i3 370M, 6Gb de memória Ram DDR3, e um SSD de 120Gb. A flag de

compilação usada, foi a O3. O sistema operacional escolhido foi o Linux Fedora, versão 22, e versão do kernel 4.0.4-301, e versão do GCC 5.1.1.

Algoritmo Sequencial

Na programação sequencial, desenvolvemos um algoritmo que calculasse os números perfeitos de 1 a 300000. Basicamente, ele faz a verificação de cada número, sendo que começa do número 1 até o número que está sendo testado, para cada número, ele testa se a divisão tem resto zero, então é um divisor natural, e consequentemente, armazena em um vetor, para depois, somar e verificar se a soma é o próprio número, logo é um número perfeito.

Algoritmo paralelo com PThread

Para implementar esta versão, o algoritmo sofreu algumas modificações para comportar variáveis globais. Assim, cada thread pode escrever nestas variáveis os valores dos números calculados.

Podemos observar aqui, que o algoritmo pode não mostrar na ordem crescente os números encontrados, pois cada thread tem sua velocidade de execução. Entretanto, isto não é um problema para nós no momento.

Algoritmo paralelo com OpenMp

Na programação usando a API OpenMp, o que podemos ressaltar é a sua facilidade de uso, uma vez que a utilização de Pthread é mais complexa. Para então realizar a implementação usando OpenMp foi mais simples, uma vez que já tínhamos o algoritmo com sua base pronta, apenas importamos a instrução ’omp’ no programa, e realizamos as demais mudanças no laço de repetição para comportar as instruções do OpenMp. Conforme Penha(2002), temos:

"A API OpenMp foi criada no início dos anos 90, a partir da idéia de embutir em Fortran e C/C++ diretivas que estendessem essas linguagens para execuções paralelas em máquinas de memória compartilhada. Isto porque os usuários destas máquinas gostariam de que fosse possível especificar através de diretivas, em programas sequenciais, quais laços de repetição poderiam ser paralelizados. Desta forma, o compilador seria responsável por paralelizar automaticamente os loops para os processadores do sistema SMP. Apesar da padronização, todas as implementações de OpenMp são similares funcionalmente, mas algumas podem divergir."

Resultados

Como podemos ver, chegamos a testar até o número 300000 em 208,120 segundos. O algoritmo sequencial, executa as instruções uma após a outra, sem nenhum paralelismo, por isso o tempo é o maior de todos.

Usando a flag -O3, tivemos uma leve diminuição do tempo, 207,84 segundos, praticamente a mesma coisa, prova que realmente o problema é definido praticamente pelo tempo de cálculo dos números maiores. A figura 2 a seguir, nos mostra o tempo de execução da versão sequencial.


Figura 2

Usando Pthread, mas com uma única thread, o tempo foi praticamente o mesmo, pois uma única thread está realizando todos os testes. Já com duas threads, podemos ver que o tempo diminuiu quase pela metade, nos mostrando a eficiência de nosso algoritmo e da paralelização em si. Usando números maiores do que 2 threads, não teve o efeito que esperávamos, pois, o processador só tem 2 cores, então ele alocou apenas 1 thread para cada core, deixando o tempo praticamente igual. Na figura 3, temos o gráfico de tempo em função do número de threads usando Pthread.

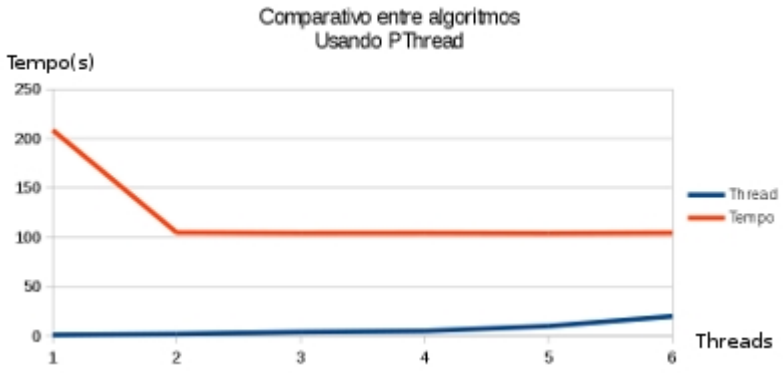


Figura 3

Em relação ao ganho de desempenho com OpenMp, pode-se observar que não houve uma grande diferença se comparando com Pthread, pois nosso equipamento só tinha 2 cores. Se analizarmos a figura 4, com até 2 threads, o resultado foi satisfatório, mas, à medida em que aumentamos o número de threads, o tempo de execução começa a aumentar também, o que nos confirma que como só temos 2 cores em nosso processador, números de threads superiores à 2 pode ser um problema, pois o escalonador do sistema operacional fica fazendo muitas trocas de contexto entre as threads, criando uma concorrência entre elas.

**Modalidade do trabalho**: Relatório técnico-científico
**Evento**: XXIII Seminário de Iniciação Científica

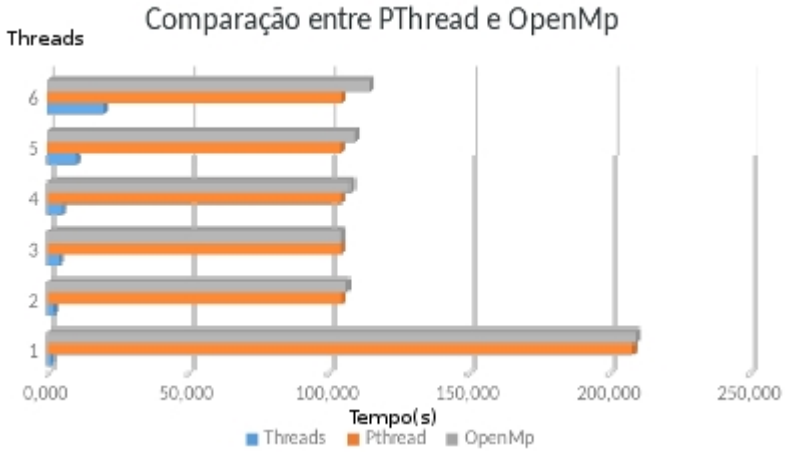


Figura 4

Conclusão

Embora a API OpenMp nos facilite a implementação paralela, deve-se ter cautela na sua utilização, pois para algum algoritmo ganha-se velocidade na programação, mas pode-se não conseguir ótimos ganhos de tempo.

Como trabalho futuro, iremos realizar a utilização da biblioteca Cuda, proprietária da nVidia, e usaremos cuda cores que estão na placa de vídeo, e que são muitos, a fim de compararmos os resultados.

Palavras chave

Números perfeitos; openmp; paralelo; sequencial; processamento paralelo

Referências

Luchetta, Valéria. "Números perfeitos", Disponível em: <http://www.matematica.br/historia/nperfeitos.html>. Acesso em: 19.jun.2015

Penha, Dulcinéia. "Análise Comparativa do Uso de Multi-Thread e OpenMP Aplicados a Operações de Convolução de Imagem" 2002, Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/wscad/2002/0018.pdf>. Acesso em: 19.jun.2015