INF2591 – Programação Concorrente: Relatório do Trabalho 2 (Bolsa de Tarefas)

Carlos Raoni de Alencar Mendes
05 de Novembro de 2012

1. Introdução

Redes de filtros, passagem de token, algoritmos de broadcast, servidores descentralizados e bolsa de tarefas são exemplos de paradigmas para realização de computações distribuídas. Estes diferentes paradigmas especificam como os diversos processos paralelos se comunicam e como o processamento é distribuído entre os mesmos. Neste trabalho é feito um estudo experimental do paradigma denominado bolsa de tarefas. Neste paradigma os processos geralmente trabalham acessando uma fila compartilhada de tarefas pendentes, obtendo e realizando tarefas desta fila ou inserindo novas tarefas na mesma.

O trabalho aplicou diversas abordagens de utilização do paradigma da bolsa de tarefas na implementação do método de cálculo de integral denominado "*Quadratura Adaptativa*". A implementação do método citado se baseou na descrição feita no trabalho [1].

Na seção 2 é feita uma breve descrição do método de *Quadratura Adaptativa*. Na seção seguinte são descritas as diversas abordagens de implementação do método citado utilizando o paradigma de bolsa de tarefas. Já na seção 3 são discutidos os experimentos e resultados obtidos no trabalho. A seção final aborda então as conclusões do trabalho.

2. Quadratura Adaptativa

O problema resolvido pelo método da *Quadratura Adaptativa* consiste em calcular a área delimitada pelo eixo x e uma função $f(x)$ não negativa em um determinado intervalo $[l, r]$, onde l e r são números reais. Desta forma o método consegue então aproximar o valor da integral de $f(x)$ neste intervalo $[l, r]$.

O método trabalha dividindo o intervalo $[l, r]$ em diversos subintervalos $[a, b]$, onde a área nestes subintervalos é aproximada através do cálculo da área do trapézio de base $b - a$ e alturas $f(a)$ e $f(b)$. O método trabalha sempre analisando um determinado subintervalo para determinar se a área do trapézio citado está dentro de um limite de tolerância com a soma das áreas dos trapézios à esquerda e a direita do ponto médio do subintervalo. Quando a tolerância é respeitada a área do subintervalo é então aproximada pela área do trapézio do subintervalo completo, caso contrário o subintervalo é divido ao meio e então a área de cada novo subintervalo é calculado separadamente utilizando a mesma abordagem.

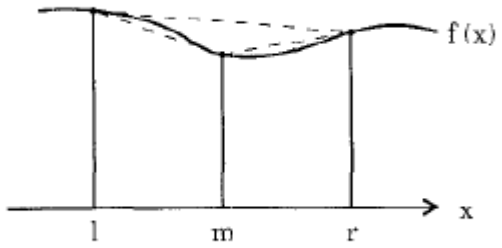


Ilustração 1. Trapézios do Método de Quadratura Adaptativa, [1].

3. Bolsa de Tarefas para o método de Quadratura Adaptativa

Como descrito na seção 1, as abordagens propostas no trabalho foram baseadas na descrição do método de *Quadratura Adaptativa* utilizando bolsa de tarefas feita em [1]. Neste trabalho, Andrews propõe que uma *thread* principal fique responsável pela criação da fila de tarefas e pela coleta dos resultados destas tarefas. As tarefas, que são nada mais que necessidades de cálculo da área de um determinado subintervalo, são realizadas pelas threads trabalhadoras, as quais comunicam seus resultados a *thread* administradora. O algoritmo é resumido na figura a seguir:

```
chan bag(a, b, fa, fb, area : real)
chan result(a, b, area : real)
Administrator:: var l, r, fl, fr, a, b, area, total : real
                other variables to record finished intervals
                fl := f(l); fr := f(r)
                area := (fl+fr)*(l+r)/2
                send bag(l, r, fl, fr, area)
                do entire area not yet computed →
                     receive result(a, b, area)
                     total := total + area
                     record that have calculated the area from a to b
                od
Worker[1:n]:: var a, b, m, fa, fb, fm : real
              var larea, rarea, tarea, diff : real
              do true → receive bag(a, b, fa, fb, tarea)
                        m := (a+b)/2; fm := f(m)
                        compute larea and rarea using trapezoids
                        diff := tarea – (larea + rarea)
                        if diff small → send result(a, b, tarea)
                        [] diff too large → send bag(a, m, fa, fm, larea)
                                            send bag(m, b, fm, fb, rarea)
                        fi
              od
```

Ilustração 2. Algoritmo de Quadratura Adaptativa utilizando Bolsa de Tarefas, [1].

Neste trabalho foram realizadas quatro implementações baseadas no algoritmo acima:

1. Cada thread recebe um intervalo fixo quando é disparada, e trabalha nesse intervalo até o fim. Ao terminar, realiza a soma de seus resultados em uma global. A thread principal espera as demais terminarem e mostra o resultado final.
2. A thread principal inicialmente cria uma lista de tarefas, contendo os extremos dos intervalos, com N tarefas. Cada thread executa uma tarefa até o final e busca uma nova tarefa nessa fila global, até que não existam mais tarefas. Nesse momento, a thread realiza a soma de seus resultados em uma global. A thread principal espera as demais terminarem e mostra o resultado final.
3. A thread principal inicialmente cria uma lista de tarefas, contendo os extremos dos intervalos, com N tarefas. Cada thread executa uma tarefa, e se ela gerar novas subtarefas, coloca uma delas na fila global e processa a outra, até que não encontre mais tarefas na fila. Nesse momento, a thread realiza a soma de seus resultados em uma global. A thread principal espera as demais terminarem e mostra o resultado final.
4. A thread principal inicialmente cria uma lista de tarefas para cada thread, contendo os extremos dos intervalos, com N/NWORKERS tarefas. Cada thread executa uma tarefa, e se ela gerar novas subtarefas, coloca uma delas em sua fila e processa a outra, até que não encontre mais tarefas na fila. Quando isso acontece, ela procura tarefas nas filas das demais threads, até não encontrar tarefas em fila alguma. Nesse momento, a thread realiza a soma de seus resultados em uma global. A thread principal espera as demais terminarem e mostra o resultado final.

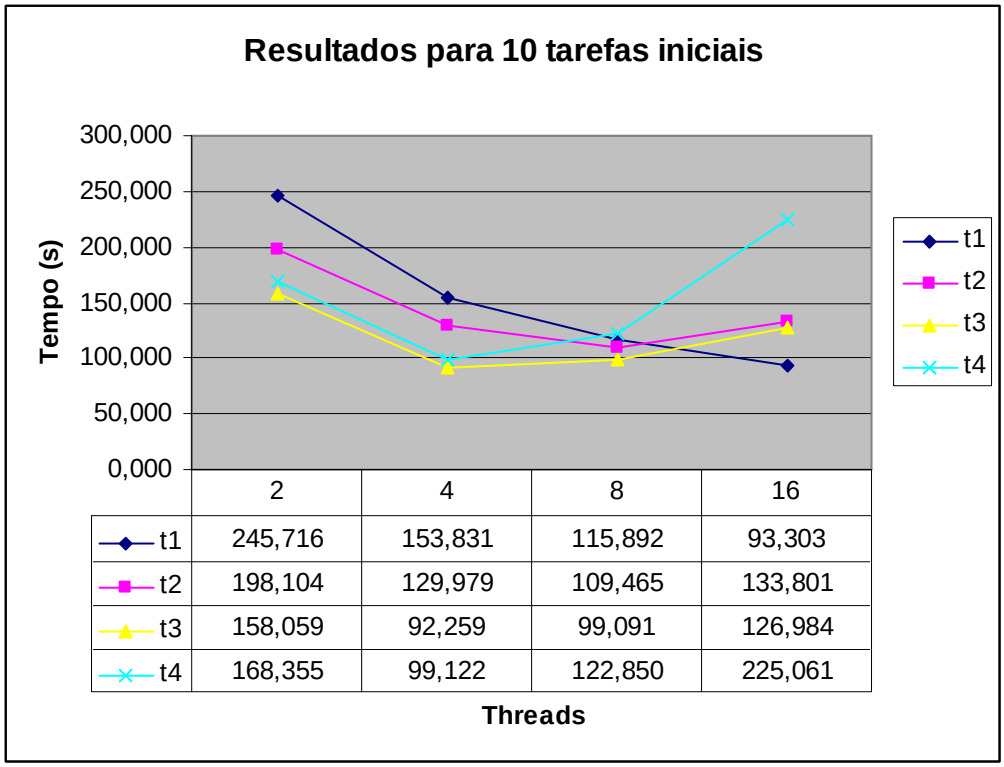
4. Experimentos e Resultados

Os programas do trabalho foram desenvolvidos na linguagem de programação C e compilados sem otimizações pelo compilador gcc disponível no pacote de software MinGW. Os experimentos foram realizados em uma máquina com 32GB de memória RAM e processador 64-bits Intel Core i3 (2.2GHz, 2 cores e 4 threads). O número de threads do processador Intel especifica quantos cores o processador consegue emular utilizando a tecnologia *HyperThreading*, então para efeito de análise dos experimentos iremos considerar a máquina como tendo 4 cores.

Foram realizados experimentos para as quatro abordagens descritas na seção anterior, variando na execução de cada abordagem o número de threads trabalhadoras (2, 4, 8 e 16) e o número de tarefas iniciais criadas (10, 100 e 1000). Na análise dos resultados foi dada prioridade para a análise da variação do número de threads trabalhadoras, logo os resultados são analisados de forma agrupada por número de tarefas iniciais criadas. A seguir são apresentados os resultados de forma agregada como descrito anteriormente.

4.1 Experimentos com 10 tarefas iniciais

A seguir é apresentado o gráfico com os resultados dos experimentos utilizando 10 tarefas iniciais, as séries são nomeadas de acordo com a numeração da descrição de cada abordagem descrita na seção anterior:



	2	4	8	16
t1	245,716	153,831	115,892	93,303
t2	198,104	129,979	109,465	133,801
t3	158,059	92,259	99,091	126,984
t4	168,355	99,122	122,850	225,061

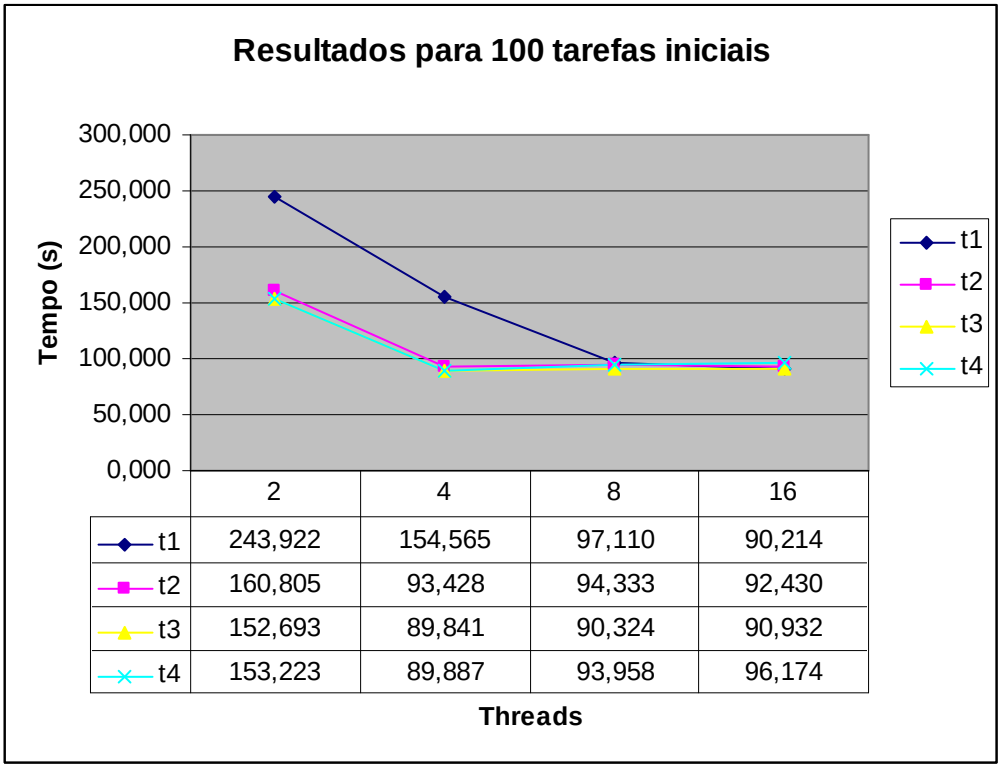
Para a primeira abordagem (*t1*) o número de tarefas iniciais é irrelevante, já que a mesma não possui fila de tarefas compartilhada. Nesta abordagem a granularidade das tarefas resolvidas por cada thread é determinada pelo número total de threads, ou seja, quanto maior o número de threads menor a granularidade (tamanho do intervalo) da tarefa. Os resultados demonstram que o aumento do número de threads e a conseqüente diminuição da granularidade das tarefas resultam em uma melhoria dos resultados de *t1*, evidenciando que uma granularidade menor de tarefa resulta em um melhor aproveitamento do paralelismo da máquina. Este mesmo resultado é observado nos gráficos dos demais experimentos, já que como dito *t1* é independente do número de tarefas iniciais, sendo *t1* incluído em cada gráfico para efeito de comparação com as outras abordagens.

Analisando os resultados de *t2* percebemos uma melhoria até o limite de 8 threads, onde existe uma inflexão de aumento de tempo computacional. Este resultado deve ocorrer por que com 16 threads temos mais threads do que tarefas a serem realizadas por cada thread de *t2*. Como cada thread em *t2* sempre requisita novas tarefas a fila compartilhada, este acesso concorrente a fila degrada o desempenho, já que pelo menos 6 threads ficarão apenas solicitando sem sucesso novas tarefas.

Os resultados para *t3* e *t4* são semelhantes a *t2*, a mudança ocorre apenas no ponto de inflexão que neste caso seria no resultado para 4 threads. Este fato deve novamente ocorrer por problemas de disputa às filas compartilhadas de tarefas, sendo mais grave em *t4*, já que na fase final de cada thread as mesmas buscam tarefas nas filas de todas as outras threads, ou seja, a disputa por recursos compartilhados é mais intensa em *t4*.

4.2 Experimentos com 100 tarefas iniciais

A seguir é apresentado o gráfico com os resultados dos experimentos utilizando 100 tarefas iniciais:

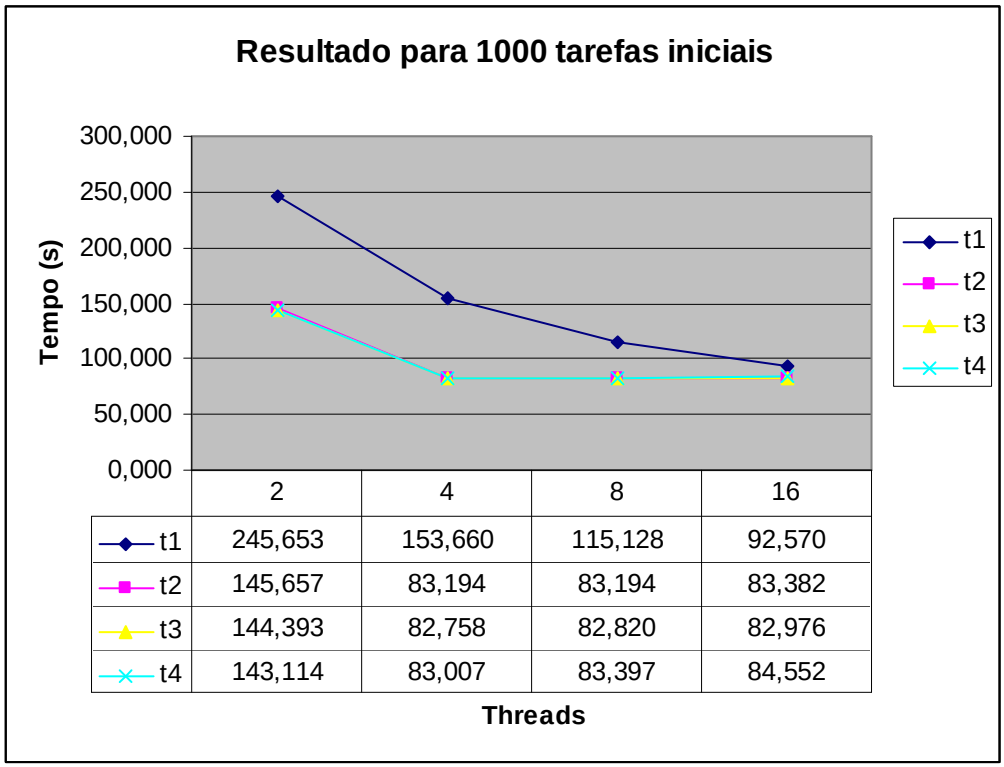


	2	4	8	16
t1	243,922	154,565	97,110	90,214
t2	160,805	93,428	94,333	92,430
t3	152,693	89,841	90,324	90,932
t4	153,223	89,887	93,958	96,174

Os resultados acima demonstram uma forte superioridade das abordagens *t2*, *t3* e *t4* sobre *t1* até 4 threads. O aumento do número de tarefas iniciais significou uma diminuição da granularidade das tarefas das abordagens *t2*, *t3* e *t4*, assim de forma conseqüente foi observada uma melhoria no desempenho das mesmas. Já observando os desempenhos para 8 e 16 threads percebemos que a granularidade das tarefas já estava em um nível adequado e portanto o aproveitamento do paralelismo da máquina foi limitado pelo número de cores do processador (4). A leve perda de desempenho novamente foi conseqüência da disputa por recursos compartilhados. O fato de termos mais tarefas implicou que menos threads ficaram ociosas apenas disputando recursos, portanto a perda de desempenho foi bem menor do que nos experimentos com 10 tarefas iniciais.

4.3 Experimentos com 1000 tarefas iniciais

A seguir é apresentado o gráfico com os resultados dos experimentos utilizando 1000 tarefas iniciais:



	2	4	8	16
t1	245,653	153,660	115,128	92,570
t2	145,657	83,194	83,194	83,382
t3	144,393	82,758	82,820	82,976
t4	143,114	83,007	83,397	84,552

Neste experimento trabalhamos com a menor granularidade de tarefas para as abordagens *t2*, *t3* e *t4*. Os resultados mostram o aprofundamento de algumas tendências com esta diminuição de granularidade para *t2*, *t3* e *t4*:

i. Melhoria geral de desempenho de *t2*, *t3* e *t4*. Como pode ser observado pelo descolamento maior em relação ao gráfico de *t1*.
ii. Aproximação dos desempenhos de *t2*, *t3* e *t4*. Como pode ser observada pela quase coincidência dos gráficos das abordagens citadas.
iii. Aproveitamento máximo do paralelismo em *t2*, *t3* e *t4* limitado pelo número de cores do processador (4). Como pode ser observada pela trajetória praticamente reta entre 4 e 16 threads.

Provavelmente a granularidade das tarefas neste experimento resultou em pouca necessidade de refinamento do cálculo dos subintervalos de cada tarefa, o que resultou na aproximação do desempenho das abordagens citadas.

5. Conclusão

Neste trabalho foram estudadas quatro abordagens de implementação paralela do método de *Quadratura Adaptativa* utilizando o paradigma de bolsa de tarefas. Os resultados demonstraram que neste paradigma o desempenho está fortemente relacionado à granularidade das tarefas. Outro fator observado como poderia esperar, é a limitação do paralelismo associado ao número de cores da máquina utilizada. A preocupação com estes dois fatores é de fundamental importância para o projeto de soluções utilizando bolsa de tarefas. Os resultados apresentados também evidenciam que com o referido paradigma é possível tirar bom proveito do potencial de paralelismo dos processadores multi-core atuais.

Referência Bibliográfica

[1] **Gregory R. Andrews**, *Paradigms for process interaction in distributed programs*, ACM Computing Surveys (CSUR), v.23 n.1, p.49-90, Março de 1991.