Impacto dos Parâmetros do StarPU no Desempenho do QR_mumps

Lucas Leandro Nesi[1], **Matheus S. Serpa**[1], **Lucas Mello Schnorr**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970, Porto Alegre – RS – Brasil

{lucas.nesi, msserpa, schnorr}@inf.ufrgs.br

Resumo. *No* runtime *StarPU e no pacote* `QR_mumps`*, existem parâmetros que podem alterar o desempenho da aplicação. Neste trabalho, são selecionados um conjunto de parâmetros de configuração de ambos os pacotes para conduzir uma análise de desempenho. A análise é dividida em duas etapas, identificado os parâmetros com maior impacto e analisado os valores para os parâmetros selecionados, apresentando a melhor configuração encontrada.*

1. Introdução

OpenMP, MPI e CUDA são exemplos de interfaces de programação paralela destinadas a computação de alto desempenho. Essas interfaces auxiliam o programador provendo uma biblioteca que implementa diversas funções úteis. Entretanto, vários aspectos importantes como a divisão de trabalho entre diferentes recursos computacionais devem ser feitos pelo programador, aumentando a complexidade do desenvolvimento. O modelo de programação baseado em tarefas permite a execução em ambientes heterogêneos, ao mesmo tempo que reduz a complexidade de programação através de um *runtime* que realiza diversas operações que seriam responsabilidade do programador.

StarPU é um *runtime* onde tarefas são submetidas dinamicamente e escalonadas para diferentes recursos tais como processadores e aceleradores [Augonnet et al. 2010]. Cada tarefa pode ter implementações para recursos diferentes como CPUs ou GPUs, e o escalonador decidirá qual recurso utilizar. O StarPU utiliza diferentes heurísticas no escalonamento, algumas mais clássicas como `LWS` (*local work stealing*), e mais sofisticadas, como o `DMDA` (*deque model data aware*) que utiliza modelos de desempenho baseado no comportamento das tarefas e transferências de dados.

Aplicações desenvolvidas utilizando StarPU podem apresentar comportamentos de desempenho diferentes dependendo dos parâmetros de configuração escolhidos durante a execução. Exemplos de parâmetros são o tamanho de bloco para subdivisão do trabalho ou a heurística adotada pelo escalonador. `QR_mumps` é uma implementação em StarPU da fatoração QR para sistemas esparsos. A implementação decompõe a matriz e aplica tarefas algébricas [Agullo et al. 2013]. O algoritmo de escalonamento, o número de trabalhadores e o tamanho do bloco na divisão da matriz são alguns dos parâmetros que interferem no desempenho dessa aplicação, tornando relevante avaliar o seu comportamento durante a execução.

Neste sentido, o objetivo deste trabalho é medir o impacto no desempenho de alguns parâmetros durante a execução da fatoração QR pelo `QR_mumps`. As contribuições deste trabalho são as seguintes. (i) Para um conjunto de parâmetros de

configuração selecionados, indicar os que possuem o maior impacto no desempenho. (ii) Considerando as opções para cada parâmetro estudado, apresentar a melhor configuração encontrada.

2. Trabalhos Relacionados

O desenvolvimento do `QR_mumps` em StarPU é apresentado em [Agullo et al. 2013]. As principais observações dos autores foram que o *runtime* produzia um máximo de 10% de sobrecarga no tempo de execução. Estudos sobre o desempenho de algoritmos utilizados na fatoração QR em blocos são apresentados em [Bouwmeester et al. 2011]. Os autores comparam o algoritmo do estado da arte, árvores de redução (utilizada no `QR_mumps`) com outros dois algoritmos desenvolvidos, observando que em alguns casos os novos algoritmos apresentavam vantagens.

O problema de encontrar a melhor configuração de parâmetros sobre o `QR_mumps` foi estudado por [Agullo et al. 2011]. Os autores, utilizando a versão somente com CPUs, estudaram os parâmetros tamanho de bloco e tamanho de bloco interno, demonstraram técnicas para reduzir a quantidade de experimentos necessários para encontrar a melhor combinação automaticamente. Diferente destes trabalhos, utilizamos a versão do `QR_mumps` sobre StarPU em ambientes heterogêneos, o que adiciona uma nova complexidade ao trabalhar com estes recursos, e utilizamos parâmetros do *runtime* para analisar o impacto no desempenho.

3. Metodologia dos Experimentos

Para avaliar o impacto no desempenho da fatoração dos parâmetros de configuração do `QR_mumps` e do StarPU, foram selecionados dois parâmetros de cada pacote como fatores. Os parâmetros são: (i) Algoritmo de escalonamento do StarPU, com valores `LWS` e `DMDA`, onde o `LWS` é um escalonador simples, e o `DMDA` mais robusto. (ii) Número de trabalhados do StarPU por GPU, com valores de zero a quatro, quatro trabalhadores por GPU significam quatro tarefas sendo executadas paralelamente utilizando aquele recurso. (iii) Tamanho do bloco na divisão da matriz no `QR_Mumps` com valores de 320, 640 e 960 . (iv) Tamanho do bloco interno, subdivisão dos blocos do `QR_Mumps`, com valores de 32 e 64. Os experimentos são divididos em duas etapas, na primeira apenas dois valores de cada fator são selecionados e é realizado o cálculo de alocação de variância com um *design* experimental $2^k n$. Na segunda etapa, os fatores mais importantes são explorados com todos os valores anteriormente descridos em um *design* fatorial completo.

As matrizes esparsas utilizadas como carga de trabalho para o `QR_mumps` foram selecionadas na SuiteSparse Matrix Collection[1], explorando diferentes tamanhos e quantidades de elementos não zeros. As matrizes com o número de linhas, colunas e elementos não zeros respectivamente são: TF16, 15437, 19321, 216173; TF17, 38132, 48630, 586218; CH7-7-b5, 35280, 52920, 211680; e MK12-b3, 51975, 13860, 207900. A máquina utilizada neste trabalho foi a Tupi, que possui uma CPU Intel Xeon E5-2620, 64 GB DDR4 de memória, e 2 Nvidia GTX 1080Ti, executando Ubuntu 18.04.1 com CUDA 10, Driver Nvidia 410 e GCC 7.3. Na primeira etapa foram realizadas 10 execuções por configuração e na segunda 30, em ordem aleatória.

[1] SuiteSparse Matrix Collection Website: `https://sparse.tamu.edu/`

4. Resultados

Os níveis dos fatores escolhidos para conduzir um design experimental $2^k n$ são os seguintes. (a) Escalonador do StarPU: `DMDA`, `LMS`; dois escalonadores com comportamentos diferentes. (b) Número de Trabalhadores por GPU: 0, 1; refletindo o uso ou não da GPU. (c) Tamanho de bloco: 320, 960; e (d) Tamanho de bloco interno: 32, 64 que representam os menores e maiores valores escolhidos. Esta etapa é limitada a matriz TF17 para reduzir o número de experimentos. Sobres os tempos de execução, é calculada a alocação de variância descrita por [Jain 1990]. Os resultados, em ordem decrescente são, fator B com 95.26% de alocação de variância, fator C 0.13%, fator D 2.93%, a combinação BC 1.34%, CD 0.14%, BD 0.10%, e BCD 0.10%. Todas as outras combinações ficaram com menos de 0.01%.

Os dados indicam que o fator A (Escalonador) não teve nenhum impacto no desempenho, incluindo todas as possíveis combinações de parâmetros utilizadas. Isso pode ser relacionado ao fato que os modelos de desempenho utilizados pelo `DMDA` não são adequados ao `QR_mumps`. O fator B (Trabalhadores por GPU) foi que teve maior impacto, chegando a 95.26%. Isso se deve aos níveis utilizados, já que zero trabalhadores significa não utilizar a GPU. O fator C (Tamanho de bloco) teve um baixo impacto, entretanto sua combinação com o fator B (BC) teve 1.34%, indicando que o desempenho do tamanho de bloco é influenciado pelo uso ou não de GPU. Por último, o fator D (tamanho de bloco interno) teve 2.93% de alocação de variância. É importante mencionar que a alocação de variância encontrada para os fatores C e D e combinações pode ter ficado mascarada, já que a utilização ou não de GPU tem muita influência no sistema.

A segunda etapa tem como objetivo encontrar a melhor configuração dos fatores estudados e é conduzida removendo o fator do escalonador, já que não teve influência na etapa um, e utilizando todos os outros níveis. Entretanto, como a utilização de GPU já mostrou seu impacto, a quantidade de trabalhadores por GPU começa a partir de um. Os resultados são apresentados na Figura 1, onde cada gráfico é uma matriz, o eixo X a quantidade de trabalhadores por GPU, o eixo Y o tempo médio com 95% de confiança.

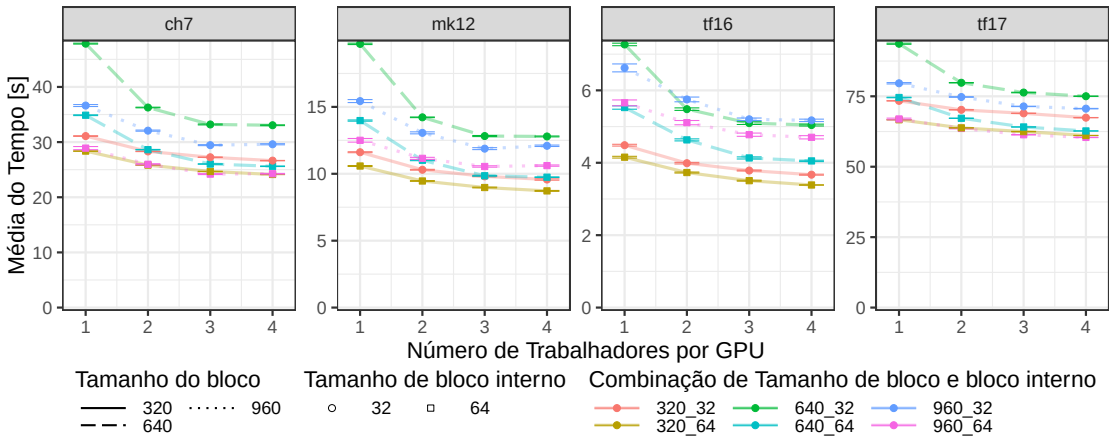

Figura 1. Média dos tempos da Fatoração QR com diferentes parâmetros.

Na Figura pode-se observar que a combinação de tamanho de bloco 320 com tamanho de bloco interno 64 é a melhor em todas as situações, de forma independente

ou empatada estatisticamente com a combinação 960 64 no casos TF17 e CH7. É interessante observar que o tamanho de bloco 960 foi melhor nas duas maiores matrizes. Em todos os casos o valor 64 para tamanho de bloco interno é o melhor. Além disso, a diminuição do tempo de execução quando aumentado o número de trabalhadores por GPU é observado em todas as quatro matrizes. Entretanto, este aumento satura-se com dois ou três trabalhadores dependendo a configuração. Desta forma, a melhor combinação em todas as matrizes foi o tamanho de bloco 320, tamanho de bloco interno 64 e pelo menos 2 trabalhadores por GPU. Enquanto a combinação 640 32 foi a pior em todas as situações.

5. Considerações Finais

Neste trabalho, foram estudados o impacto de um conjunto de parâmetros de configuração no desempenho da fatoração QR no pacote `QR_mumps` utilizando o StarPU. Foram selecionados dois parâmetros de cada pacote, e quatro matrizes como cargas de trabalho. Os experimentos foram divididos em duas partes, primeiramente entender o impacto utilizado a técnica de alocação de variância em um *design* experimental $2^k n$ e após limitar os parâmetros uma segunda etapa experimental utilizando todos os níveis de valores possíveis utilizando um design fatorial completo.

Dentre os parâmetros estudados, ficou observado que o escalonador do StarPU não apresentou impacto no desempenho. No caso do tamanho de bloco interno, o melhor valor encontrado em todos os casos foi de 64. No caso do tamanho de bloco, a opção de 320 teve os melhores resultados em todos os casos de forma isolada, ou empatada estatisticamente com o valor de 960, que empatou nas duas maiores matrizes. Em relação ao número de trabalhadores por GPU, foi notado um ganho no desempenho quando aumentado de um trabalhador para dois, entretanto aumentos acima deste valor não apresentaram ganhos estatísticos em todos os casos.

Trabalhos Futuros incluem o estudo das causas destes comportamentos, podendo ser utilizados os rastros gerados pelo StarPU, e a investigação de outros parâmetros que possivelmente podem interferir no desempenho.

Referências

Agullo, E., Buttari, A., Guermouche, A., and Lopez, F. (2013). Multifrontal qr factorization for multicore architectures over runtime systems. In *Euro-Par 2013 Parallel Processing*, Berlin, Heidelberg. Springer Berlin Heidelberg.

Agullo, E., Dongarra, J., Nath, R., and Tomov, S. (2011). A fully empirical autotuned dense qr factorization for multicore architectures. In *Euro-Par 2011 Parallel Processing*, Berlin, Heidelberg. Springer Berlin Heidelberg.

Augonnet, C., Clet-Ortega, J., Thibault, S., and Namyst, R. (2010). Data-Aware Task Scheduling on Multi-Accelerator based Platforms. In *16th Intl. Conference on Parallel and Distributed Systems*, Shangai.

Bouwmeester, H., Jacquelin, M., Langou, J., and Robert, Y. (2011). Tiled qr factorization algorithms. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 7:1–7:11, New York, NY, USA. ACM.

Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.