

Implementações Eficientes da Heurística Min-Min para o HCSP em GPU

Rafael F. Schmid¹, Edson N. Cáceres¹

¹Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Campo Grande, MS – Brasil

rafaelfschmid@gmail.com, edson@facom.ufms.br

***Abstract.** Min-min heuristic is one of the most indicated algorithm to solve HCSP, when we are looking for performance. A previous result presented an efficient implementation to this problem with asymptotic complexity $O(t \log(t)m)$. In this work, we implement this problem with linear complexity. Our implementation is competitive with the previous results.*

1. Introdução

A utilização de grades computacionais e computação em nuvem têm obtido bons resultados na solução de problemas complexos. De uma forma geral, grades computacionais e computação em nuvem são compostas de vários computadores heterogêneos que são capazes de trabalhar de forma cooperativa.

Considerando que temos várias tarefas a serem executadas num conjunto heterogêneo de computadores, onde o tempo de execução de uma tarefa pode variar de acordo com o computador a ser utilizado, temos que resolver o problema de como escalonar as tarefas para serem resolvidas nos computadores no menor tempo possível. Nesses casos, temos uma competição entre as tarefas para utilizar o computador que seja capaz de resolvê-las no menor tempo [Nesmachnow and Canabé 2011]. Esse problema é denominado o Problema de Escalonamento em Ambientes de Computação Heterogênea (HCSP - Heterogeneous Computing Scheduling Problem).

Como trata-se de um problema NP-completo, muitas heurísticas tem sido aplicadas para resolvê-lo, como min-min, max-min, Algoritmos Genéticos (GA), *Genetic Simulated Annealing* (GSA), *Sufferage*, etc [Braun et al. 2001]. Em função de suas características, a heurística min-min se mostrou como uma das mais rápidas para esse problema e apresenta soluções quase tão boas quanto GA, que é uma das heurísticas mais demoradas [Wu et al. 2000].

Nesse trabalho, baseado na implementação híbrida de [Pedemonte et al. 2016] e no trabalho de [Schmid et al. 2016], desenvolvemos uma implementação usando a heurística min-min para o HCSP. Nosso resultado tem complexidade linear e a sua implementação se mostrou competitiva em relação aos resultados anteriores.

2. O problema do HCSP e a Heurística Min-Min

No modelo apresentado nesse trabalho, todas as tarefas podem ser executadas independentemente, sem se preocupar com a ordem de execução, e quando uma tarefa inicia ela é executada até o final. As aplicações a serem executadas são compostas por uma

coleção de tarefas indivisíveis, que não tem dependência entre elas, comumente chamadas de *metatask*. Problemas de escalonamento consistem em tentar minimizar o tempo gasto para executar todas essas tarefas. A formalização abaixo apresenta o modelo matemático para o HCSP com o intuito de minimizar o *makespan* (tempo gasto a partir do momento em que a primeira tarefa começa até o momento em que a última é completada [Nesmachnow and Canabé 2011]):

- Dado um sistema de computação heterogênea composto de um conjunto de M máquinas $P = m_1, m_2, \dots, m_M$ e uma coleção de N tarefas $T = t_1, t_2, \dots, t_N$ a serem executadas.
- Considere uma função de tempo de execução $ET : P \times T \rightarrow R^+$, onde $ET(t_i, m_j)$ é o tempo necessário para executar a tarefa t_i na máquina m_j .
- O objetivo do HCSP é encontrar uma atribuição de tarefas às máquinas (uma função $f : T^N \rightarrow P^M$) que minimize o *makespan*, conforme definido na Equação 1.

$$\max_{m_j \in P} \sum_{\substack{t_i \in T \\ f(t_i)=m_j}} ET(t_i, m_j) \quad (1)$$

A heurística min-min começa com o conjunto U de todas as tarefas não mapeadas. Então, o conjunto de máquinas de menor tempo de conclusão M para cada tarefa em U é encontrado. A tarefa cujo tempo de conclusão seja o menor, é escolhida e atribuída à máquina correspondente. A nova tarefa mapeada é removida do conjunto U , e o processo se repete até que todas as tarefas sejam mapeadas [Braun et al. 2001].

3. Trabalhos Relacionados

Nesmachnow e Canabé (2011) apresentaram uma implementação em GPU para a heurística min-min e obtiveram *speedups* de aproximadamente seis vezes em relação a uma implementação sequencial. Ezzatti et al. (2013) propuseram um algoritmo min-min, que chamamos de **min-min sort**, que efetua a ordenação segmentada sobre os tempos que cada máquina leva para executar cada uma das tarefas, e depois verifica, nessa ordem, as tarefas de cada máquina, selecionando a menor para aquela iteração. Se utilizado um algoritmo de ordenação linear, esse algoritmo alcança complexidade assintótica também linear: $O(tm)$, onde t é o número de tarefas e m o número de máquinas, mas quando da implementação os autores utilizaram um algoritmo $O(t \log(t)m)$. Posteriormente, Pedemonte et al. (2016) apresentaram uma implementação híbrida GPU-CPU do *min-min sort* com resultados melhores que os anteriores.

4. Solução Proposta e Resultados Experimentais

Os experimentos deste artigo foram realizados em uma máquina com o Sistema Operacional Ubuntu 16.04, 16 GB de memória RAM, processador Intel Coretm i5-4460 CPU @ 3.20 GHz x 4 e placa gráfica GeForce GTX 1050 Ti com 4 GB de memória, e os cenários de teste utilizados foram obtidos do endereço <https://par-cga-sched.gforge.uni.lu/instances/etc/>.

Baseado no trabalho de [Schmid et al. 2016], apresentamos uma implementação da heurística *min-min sort* utilizando o *radix-sort*. O principal objetivo é utilizar uma

ordenação segmentada que garanta complexidade linear para a heurística min-min. Para efetuar as comparações também foi implementada: a min-min padrão em CPU e GPU; e a *min-min sort* em CPU, GPU e Híbrida (GPU-CPU).

A implementação em GPU da heurística min-min padrão foi testada utilizando duas estruturas diferentes. A primeira utiliza as linhas da matriz para representar as máquinas e as colunas para representar as tarefas. Cada thread CUDA é responsável por processar uma tarefa, ou seja, encontrar a máquina em que a tarefa possui o menor tempo de conclusão. Cada thread armazena seu resultado em uma posição do vetor resultante, onde é realizada uma redução paralela para encontrar a tarefa que deve ser escalonada naquela iteração. Dessa forma, as threads acessam a mesma linha da matriz a cada iteração (Figura 4), favorecendo a coalescência. Ao inverter essa estrutura perdemos a coalescência e os tempos sobem bastante, por isso optamos não abordá-la aqui.

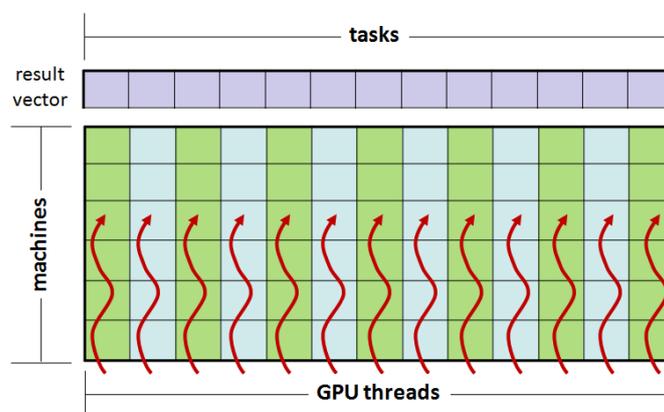


Figura 1. Estratégia de implementação em GPU da heurística min-min [Nesmachnow and Canabé 2011].

A implementação paralela do *min-min sort* de [Ezzatti et al. 2013], efetua a ordenação segmentada utilizando uma biblioteca CUDA, e depois utiliza uma redução paralela para identificar as máquinas que possuem o menor tempo de conclusão na iteração corrente. Para essa estratégia representamos as máquinas em linhas e as tarefas em colunas, estrutura necessária para efetuarmos a ordenação segmentada de forma eficiente.

A nossa implementação do *min-min sort* híbrido CPU-GPU usa a mesma estrutura da versão paralela do algoritmo *min-min sort*. Na primeira etapa, a ordenação das linhas da matriz, é efetuada em GPU. Posteriormente, os dados são copiados de volta para CPU e a mesma implementação da versão sequencial do *min-min sort* é executada. Para a ordenação utilizamos a biblioteca CUB (*radix-sort*).

A Figura 2 apresenta o *speedup* de cada uma das implementações: baseadas no *merge-sort* (Pedemont et al.) e baseadas no *radix-sort* (implementação proposta) quando comparadas com a implementação sequencial do min-min padrão. Com isso, podemos observar que, apesar de estarmos utilizando um algoritmo linear, em termos de desempenho, o comportamento é semelhante ao algoritmo de complexidade $O(t \log(t)m)$ da biblioteca MGPU.

Podemos notar também, que a implementação sequencial do *min-min sort* apresenta melhores resultados para matrizes menores, já que o acesso não é coalescido, devido

a ordenação segmentada do início do algoritmo. O motivo de apresentar melhores resultados é que a ordenação segmentada em GPU possui desempenho superior à ordenação sequencial, prova disso são os desempenhos bem melhores das implementações híbridas.

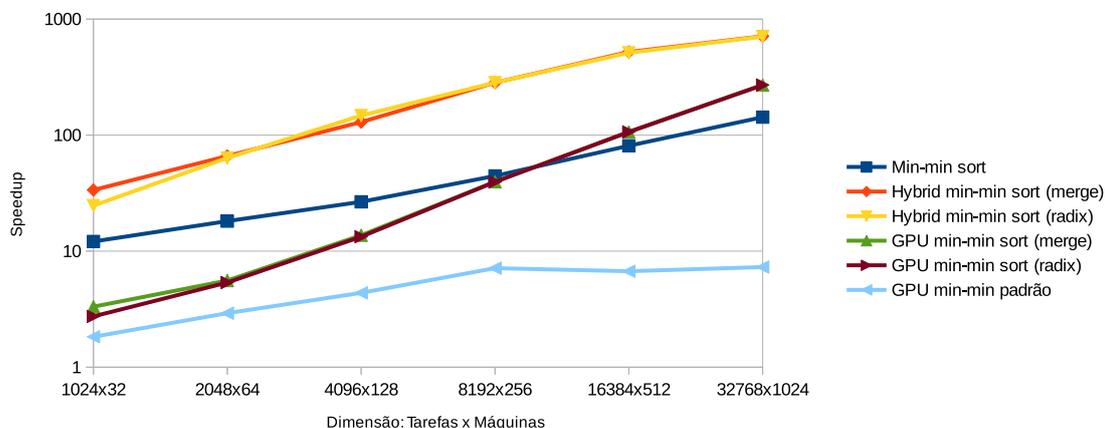


Figura 2. Speedup das implementações do Min-Min.

5. Conclusões

A implementação que apresentamos possui complexidade linear e os resultados obtidos na GPU e GPU-CPU são competitivos.

Referências

- Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., and Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810 – 837.
- Ezzatti, P., Pedemonte, M., and Martín, Á. (2013). An efficient implementation of the min-min heuristic. *Computers & Operations Research*, 40(11):2670–2676.
- Nesmachnow, S. and Canabé, M. (2011). Gpu implementations of scheduling heuristics for heterogeneous computing environments. In *XVII Congreso Argentino de Ciencias de la Computación*.
- Pedemonte, M., Ezzatti, P., and Martín, Á. (2016). Accelerating the min-min heuristic. In *Parallel Processing and Applied Mathematics*, pages 101–110. Springer.
- Schmid, R., Pisani, F., Borin, E., and Cáceres, E. (2016). An evaluation of segmented sorting strategies on gpus. In *IEEE 18th International Conference on High Performance Computing and Communications (HPCC)*, pages 1123–1130. IEEE.
- Wu, M.-Y., Shu, W., and Zhang, H. (2000). Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 375–385. IEEE.