Uma interface de programação paralela utilizando C++11

Daniel Di Domenico, Gerson Geraldo H. Cavalheiro

[1]Universidade Federal de Pelotas (UFPEL)
Programa de Pós-Graduação em Computação
Pelotas – RS – Brasil

{ddomenico,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *O presente trabalho propõe uma interface de programação paralela de alto nível utilizando C++11 que visa oferecer suporte a diferentes tipos de arquiteturas. Através da divisão dos conceitos de tarefa, thread e escalonamento, a interface pretende permitir, além de heterogeneidade, a implementação de um escalonador customizado sem exigir alterações na estrutura do código a ser executado em paralelo.*

1. Introdução

Processadores multicore e aceleradores estão sendo utilizados em diversas soluções na área da Programação de Alto Desempenho (PAD), o que gerou desafios na implementação das aplicações a fim de explorá-los de forma eficiente. Alguns dos problemas que precisam ser enfrentados neste processo são a exposição da concorrência e o acesso a dados compartilhados, além do escalonamento e sincronização de tarefas.

Visando simplificar o desenvolvimento de aplicações paralelas, algumas iniciativas propuseram modelos de programação onde apenas a concorrência do programa é exposta ao desenvolvedor, sendo seu paralelismo explorado pelos recursos da própria ferramenta (tais como OpenMP, TBB e Cilk). Implementações que seguem este conceito são denominadas $n \times m$, onde uma aplicação, empregando um escalonador, mapeia as n tarefas por ela geradas para as m *threads* presentes no sistema. Dentro desta ideia, apenas a expressão da concorrência deve ser explícita na aplicação (criação das tarefas), enquanto o mapeamento destas tarefas para o *hardware* ocorre de forma implícita.

Seguindo a definição de implementação $n \times m$, este trabalho tem por objetivo propor uma interface de programação paralela de alto nível em C++ na forma de biblioteca voltada ao processamento heterogêneo (CPU, GPU e CPU+GPU). Para alcançar este propósito, serão utilizadas técnicas de metaprogramação presentes no C++11. Apesar de já existirem modelos de programação C++ semelhantes ao aqui proposto (como [Heller et al. 2017] e [Edwards et al. 2012]), este projeto visa desassociar completamente a interface de programação do mecanismo de escalonamento, que poderá ser implementado de forma customizada sem requerer alterações no código a ser executado em paralelo.

2. Interface de programação paralela proposta

A interface proposta será modelada de modo a dividir de forma clara os três conceitos existentes em uma aplicação paralela: **tarefa**, ***thread*** e **escalonamento**. A concorrência da aplicação é definida em termos de **tarefas**, que após implementadas são submetidas a execução por unidades de processamento representadas por ***threads***. Por fim, o **escalonador** é o meio utilizado para coordenar as comunicações entre as tarefas e as *threads*.

A Figura 1 detalha como planeja-se proceder a implementação da interface. Ela encapsulará os conceitos de tarefa, *thread* e escalonador em estruturas abstratas, as quais poderão ser estendidas a fim de modelar a concorrência do programa.

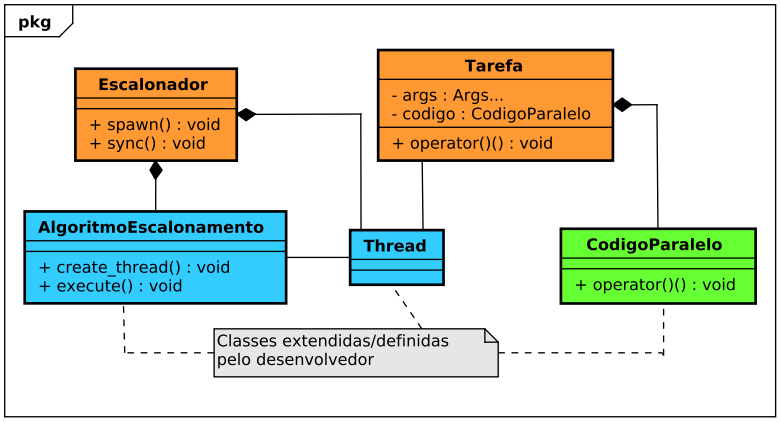

Figura 1. Diagrama de classes com a estrutura da interface proposta

De acordo com a Figura 1, propõem-se que o código a ser processando de forma paralela seja mapeado como um objeto C++ (lambdas ou *functors*) na classe `CodigoParalelo`. Para ser executado, ele deverá ser vinculado a uma `Tarefa`. A classe `Tarefa` possuirá recursos para permitir a passagem de parâmetros tipados ao código, garantindo segurança de tipos (recurso possível por meio da metaprogramação).

Assim que definida a `Tarefa`, a classe `Thread` poderá ser estendida a fim de determinar como e onde ela será executada. Uma `Thread` será uma abstração de uma unidade de processamento, sendo utilizada para mapear os recursos de hardware da plataforma. A `Thread` separa totalmente a implementação da tarefa do meio empregado para executá-la, viabilizando o suporte a diferentes tipos de arquiteturas. Com esta modelagem, uma mesma tarefa poderá ser executada por uma `Thread` de CPU e outra de GPU, disponibilizando suporte para o processamento heterogêneo.

Por fim, a classe `AlgoritmoEscalonamento` possibilitará criar uma heurística customizada (escalonamento dinâmico e independente do código da tarefa) para gerenciar a execução da aplicação. Após defini-la, as chamadas para o método `spawn()` e `sync()` iniciarão e sincronizarão o processamento de uma `Tarefa` em uma `Thread`.

3. Trabalhos futuros

Na continuidade deste trabalho, planeja-se implementar a interface, inicialmente com recursos para a execução de aplicações em ambiente multicore. Na sequência, será implementado o suporte a GPUs e ao processamento heterogêneo.

Referências

Edwards, H. C., Sunderland, D., Porter, V., Amsler, C., and Mish, S. (2012). Manycore performance-portability: Kokkos multidimensional array library. *Scientific Programming*, 20(2):89–114.

Heller, T., Diehl, P., Byerly, Z., Biddiscombe, J., and Kaiser, H. (2017). HPX – An open source C++ Standard Library for Parallelism and Concurrency. In *Proceedings of OpenSuCo 2017, Denver, Colorado USA, November 2017 (OpenSuCo17)*, page 5.