Metalinguagem para processamento de alto desempenho em arquiteturas híbridas

André Leon Sampaio Gradvohl[1]

[1]Faculdade de Tecnologia – Universidade Estadual de Campinas
(FT–UNICAMP)
R. Paschoal Marmo, 1888 – CEP 13484-332 – Jd. Nova Itália – Limeira – SP – Brasil

gradvohl@ft.unicamp.br

1. Introdução

Há problemas computacionais que requerem muito poder computacional para serem resolvidos em tempos plausíveis. Por exemplo, previsões meteorológicas são problemas computacionalmente intensos onde a resposta deve ser obtida o mais rápido possível, sob pena de ter uma resposta após o evento que se deseja prever.

Para atender à demanda por um processamento de alto desempenho (PAD), existem diferentes modelos de arquiteturas de computadores. De um modo geral, do ponto de vista do sistema de memória, existem os sistemas com memória distribuída e com memória compartilhada. Nos sistemas de memória distribuída, diversos processadores – cada um com seu próprio sistema de memória – são interligados por uma rede de comunicação. Cada um desses módulos, formado por processador e memória, trata um pedaço de uma tarefa maior. Ao longo do tempo, trocando informações entre si, esses módulos podem produzir resultados intermediários que, após combinados, podem convergir para um resultado final. Nos sistemas de memória compartilhada, por sua vez, os diversos processadores acessam um único bloco de memória. Com isso, a troca de informações para a solução do problema ocorre na própria memória, reduzindo assim o problema da latência.

Para cada um desses sistemas há um modelo de programação adequado. No caso de sistemas com memória distribuída, modelos de programação baseados em trocas de mensagens entre processos remotos são os mais indicados. Nesse sentido, a interface de programação de aplicações (API) *Message Passing Interface* (MPI) é o padrão *de facto*. Por outro lado, os sistemas de memória compartilhada podem usar funções do próprio sistema operacional – especialmente as primitivas de sincronização – para orquestrar o acesso à memória pelos diversos processadores. Nesse caso, as APIs OpenMP e PThreads são as mais utilizadas para esse tipo de sistemas.

Contudo, atualmente, existem computadores para PAD que combinam ambos os sistemas de memória e são chamados de sistemas de arquiteturas híbridas [Rabenseifner 2003]. Para esse tipo de sistema, dois ou mais modelos de programação paralela devem ser combinados para usufruir ao máximo do poder computacional disponível [Andrijauskas and Gradvohl 2012]. É importante ressaltar que os sistemas híbridos são diferentes do modelo chamado "Espaço de Endereçamento Global Particionado" (*Partitioned Global Address Space* – PGAS). No PGAS há um espaço de endereçamento de memória global, particionado logicamente, no qual cada uma dessas partições é alocada a um processo ou *thread*. Nesse caso, cada partição do espaço de memória deve ter afinidade com um processo ou thread particular para explorar o conceito de localidade de

referências [Coarfa et al. 2005]. No entanto, para usar esse modelo é preciso utilizar linguagens de programação específicas como *Unified Parallel C* (UPC), Coarray Fortran, X10 ou Chapel por exemplo.

Para um programador experiente, a adaptação para um ou mais desses modelos de programação não é tão difícil. A questão é que, na maior parte das vezes, o desenvolvimento de software para sistemas híbridos é feita por profissionais com forte formação nas chamadas ciências duras – e.g. biologia, física, química – e menos em programação. Em outras palavras, são profissionais que se concentram muito mais na solução analítica do problema do que na solução computacional.

2. Proposta de solução do problema

Dadas tais circunstâncias e considerando a necessidade de um modelo de programação que permita a implementação de software para arquiteturas híbridas, o trabalho de pesquisa descrito neste texto e ora em andamento propõe a especificação de uma metalinguagem para PAD nessas arquiteturas e seu respectivo pré-compilador. Neste contexto, uma metalinguagem de programação é um conjunto de anotações que, depois de analisadas em um texto de programa, se transformarão em sequências de chamadas de funções nas diversas APIs (MPI, PThreads ou OpenMP), conforme ilustra a Figura 1.

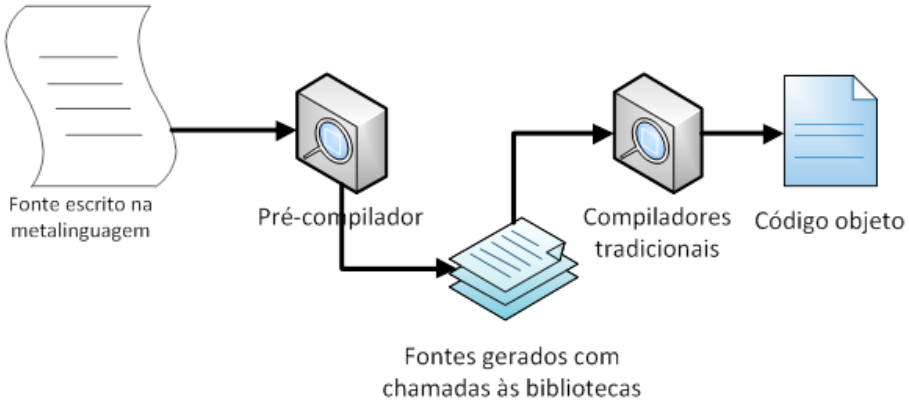

Figura 1. Arquitetura do metacompilador.

Dessa forma, o programador que usar tal metalinguagem não precisará se preocupar em aprender a sintaxe de várias APIs. O programador pode se concentrar muito mais no seu modelo analítico, sem deixar de considerar as características das arquiteturas híbridas na solução do problema. A Figura 2-(a) mostra um exemplo de código fonte original com suas respectivas anotações; e a Figura 2-(b) mostra o resultado da pré-compilação, com as anotações substituídas pelas respectivas chamadas de funções nas APIs MPI e OpenMP.

3. Metodologia

No desenvolvimento do projeto identificou-se as estruturas mais usuais de programação em cada uma das APIs. Entre elas destacam-se as estruturas para distribuição de tarefas e estruturas para sincronização.

Para cada tipo de estrutura, foram estabelecidas anotações que indicam o tipo de ação a ser tomada e o respectivo contexto (memória distribuída ou compartilhada).

```
#include <stdio.h>
#include "hysop.h"

int main(void)
{
  @initialize

  @define tasks(3)
  {
    printf("ola mundo!");
  }

  @finalize
  return 0;
}
```

(a) Código Original

```
#include <stdio.h>
#include <mpi.h>
#include <omp.h>

int main(void)
{
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

  omp_set_num_threads(3);

  #pragma omp parallel
  {
    printf("ola mundo!");
  }

  MPI_finalize();
  return 0;
}
```

(b) Código gerado pelo metacompilador

Figura 2. Código fonte anotado e respectivo código fonte gerado pelo metacompilador.

Por definição, as anotações começam sempre com o símbolo @. Em seguida, iniciou-se a construção de um metacompilador que será responsável pela análise e tradução das anotações nas respectivas chamadas às APIs.

4. Resultados Esperados

O projeto encontra-se em uma fase de especificação da metalinguagem. Por isso, ainda não há resultados significativos. No entanto, espera-se que a metalinguagem proposta ajude no desenvolvimento de software para sistemas híbridos, com maior velocidade e qualidade. Este projeto é financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Projeto 2011/00861-0.

Referências

Andrijauskas, F. and Gradvohl, A. L. S. (2012). Solar filaments detection using parallel programming in hybrid architectures. In *Proceedings of the 2012 workshop on High-Performance Computing for Astronomy Date*, Astro-HPC '12, pages 41–48, New York, NY, USA. ACM.

Coarfa, C., Dotsenko, Y., Mellor-crummey, J., Cantonnet, F., El-ghazawi, T., Mohanty, A., and Yao, Y. (2005). An evaluation of global address space languages: Co-array fortran and unified parallel c. In *Principles and Practice of Parallel Programming*, pages 36–47.

Rabenseifner, R. (2003). Hybrid parallel programming on hpc platforms. In *5th European Workshop on OpenMP*, pages 185–194.