Uma Nova Abordagem para Programação Visual de Aplicações Paralelas de Alto Desempenho

André Ricardo[1], André Pereira[1], David Duarte[1], João Abreu[1], Guilherme Esmeraldo[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE) – *campus* Crato
Crato – CE – Brazil

{arbezerra,andrepereiraleite}@gmail.com,
{davidduarte,joaoalberto,guilhermealvaro}@ifce.edu.br

Resumo. *A computação paralela é uma técnica que busca dividir as tarefas entre os processadores para aumento de desempenho do sistema. Contudo, desenvolver novas aplicações paralelas não é uma tarefa simples, pois deve-se identificar os pontos de paralelismo da aplicação e expressá-los em linguagens de programação sequenciais. Este artigo apresenta uma nova abordagem que permite a modelagem visual e, com isso, é possível abstrair o paralelismo da aplicação além de gerar automaticamente o código necessário à paralelização. Resultados preliminares mostram que, com a utilização da ferramenta proposta, é possível aumentar a produtividade no desenvolvimento de novas aplicações paralelas de alto desempenho.*

1. Introdução

Com o desenvolvimento das tecnologias de integração de circuitos, os processadores passaram a ficar mais complexos [BOUAJILA et al., 2011], sendo muito comum encontrar processadores com 2 ou mais núcleos para processamento paralelo. Nesse contexto, se desenvolveu a programação paralela, a qual é uma técnica que se caracteriza pela divisão de tarefas entre vários processadores, obtendo-se, desta forma, um ganho de desempenho na sua execução [GEBALI, 2011].

A criação de aplicações paralelas não é uma tarefa simples e exige certo nível de experiência dos programadores [MATTSON, SANDERS E MASSINGILL, 2005]. Assim, diversas técnicas foram criadas para auxiliar o desenvolvimento de novas aplicações. A maioria baseia-se na técnica de Dividir para Conquistar, onde o princípio básico é dividir o problema em subproblemas e solucioná-los individualmente, agrupando as soluções, até compor a solução do problema [CORMEN et al., 2012]. Baseadas nela, estão as técnicas de divisão funcional e divisão de domínio [PACHECO, 2011]. Na divisão funcional, ou *decomposição em pipeline*, o programa paralelo tem suas tarefas divididas entre processos para resolver o problema de forma colaborativa. Na divisão por domínio, ou *decomposição de dados*, os dados do problema são divididos entre processos, de forma que eles realizam a mesma quantidade de operações nos seus respectivos conjuntos de dados. Ao fim, os resultados são correlacionados de forma que façam sentido para a solução do problema [PALACH, 2014]. Mattson, Sander e Massingill (2005) citam que o desenvolvimento de uma nova aplicação paralela se resume em quatro fases: exposição dos pontos de concorrência no problema,

estruturação do algoritmo, projeto do programa paralelo e implementação do programa paralelo, utilizando ambientes de programação paralela, como MPI. Contudo, o suporte desses ambientes é inerente ao uso de uma linguagem de programação e, segundo Lee e Webber (2003), há dois problemas básicos: o primeiro refere-se à linearidade do código fonte, não permitindo uma representação intuitiva dos fluxos paralelos de controle; e complexidade de paralelismo, onde o programador deve identificar e controlar o paralelismo, o compartilhamento de dados, protegendo o acesso aos recursos compartilhados e garantir que as seções críticas do código sejam devidamente delimitadas.

Lee e Webber (2003) fazem ainda um apelo ao uso de "linguagens visuais", pois elas permitem representar fluxos paralelos e sua sintaxe pode ser adaptada para implicitamente abordar várias questões relacionadas ao paralelismo e controle de concorrência. Trabalhos, como CODE 2.0 [NEWTON e BROWNE, 1992] e KAIRA [BÖHM et al., 2014], apresentam ambientes para a criação visual de novas aplicações paralelas. Nesses trabalhos, as aplicações são criadas com a utilização de modelos de grafos, os quais, segundo Newton e Browne (1992), permitem expressar o paralelismo em alto nível de abstração, onde os arcos podem representar fluxos de trocas de dados entre os vértices, e estes podem representar computações básicas sequenciais.

Este artigo apresenta uma nova abordagem para suporte a criação visual de novas aplicações paralelas: inclui suporte a um novo modelo de grafo direcionado, onde o conjunto de vértices podem representar recursos compartilhados (conjuntos de dados de entrada e saída) e computações básicas (sequenciais ou paralelas). As arestas representam fluxos de trocas de dados entre esses elementos.

O restante do artigo é dividido da seguinte maneira: Na próxima seção, apresenta-se o modelo de grafo bem como o modelo de paralelização propostos. A Seção 3 apresenta um estudo de caso e alguns resultados preliminares. Por fim, as conclusões e trabalhos futuros são demarcados na Seção 4.

2. Modelos Propostos para Programação Visual

Esta seção é subdividida em duas partes: inicialmente, apresenta-se o modelo de grafo proposto para modelagem visual de novas aplicações paralelas; e, em seguida, é descrito o modelo de paralelização adotado.

2.1. Modelo de Grafo

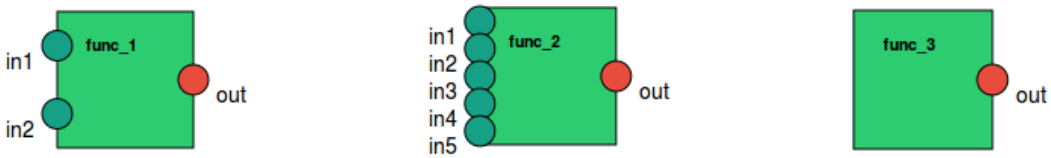
Neste trabalho, a estrutura utilizada para modelar aplicações paralelas é um grafo direcional [CORMEN, 2012], cujos vértices podem representar elementos de dados e elementos de processamento, bem como as arestas representam os fluxos de dados entre elementos de dados e de processamento.

Um vértice de elemento de dados pode consistir de variável, estrutura de dados e arquivo em disco. Um elemento de dados pode ser de entrada (dados a serem processados) ou de saída (dados resultantes de processamento). A Figura 1 apresenta dois exemplos de vértices de elementos de dados, sendo um de entrada (a) e outro de saída (b), respectivamente.

(a) (b) (c)

Figura 1. Exemplos de vértices de entrada (a) e saída (b), e fluxo de dados (c).

O fluxo de dados entre os vértices do modelo de grafo adotado é dado no sentido da esquerda para direita. Com isso, vértices com interface à direita, circulo vermelho no vértice da Figura 1 (a), por exemplo, consistem de vértices de dados de entrada. Já na Figura 1 (b), a interface à esquerda, círculo em cor verde, denota que o vértice é de saída, ou seja, receberá dados de outro vértice. A figura a seguir ilustra o fluxo de troca de dados entre vértice de entrada e de saída. Na Figura 1(c), a aresta, que liga o vértice de entrada (à esquerda) ao vértice de saída (à direita), indica que os dados da entrada serão enviados para a saída. Os vértices de processamento incluem as instruções necessárias ao processamento de dados, que podem ser computações (funções) básicas sequenciais ou paralelas. A Figura 2 ilustra três vértices representando funções com diferentes números de interfaces.



def func_1 (in1, in2): def func_2 (in1, in2, in3, in4, in5): def func_3 ():

(a) (b) (c)

Figura 2. Exemplos de vértices de processamento com diferentes interfaces.

Na Figura 2 (a), temos um vértice com duas interfaces de entrada de dados. Logo abaixo do vértice, temos a definição da respectiva função. Nela, vemos que os parâmetros de entrada (in1 e in2) correspondem às interfaces de entrada de dados do vértice. Nas Figuras 2 (b) e (c), temos vértices com cinco e nenhuma interface de entrada, respectivamente.

2.2. Modelo de Paralelização

Neste trabalho, as funções representadas pelos vértices são paralelizadas utilizando o *Padrão de Projetos Decorator* [GAMMA et al., 2000]. Os *decorators* propostos se baseiam na técnica de divisão de domínio (divisão do conjunto de dados em subconjuntos de mesmos tamanhos para serem processados em paralelo). Com isso, uma função definida em um vértice de processamento pode ser aplicada a cada um dos subconjuntos gerados com o *decorator*. Além disso, o projetista da aplicação paralela pode escolher entre processos e threads para realizar as tarefas de processamento paralelo (aplicação das funções aos subconjuntos do problema). Após o processamento dos subconjuntos, as soluções locais são reagrupadas, através de uma função de *callback*, para compor a solução final. Por exemplo, a definição da função soma, em Python, acrescida de um *decorator* com processamento paralelo por 4 processos, poderia ser dada por:

```
1.  @process(4, sum)          # decorator
2.  def soma(in_1, in_2):     # definicao da funcao
```

Figura 3. Função soma "decorada" com suporte a processamento paralelo.

Na Figura 3, com o uso do *decorator* proposto (linha 1), cada uma das entradas de dados da função soma (in_1 e in_2, declaradas na linha 2) será dividida em 4 subconjuntos. Em seguida, a função *soma* será aplicada a cada par dos subconjuntos por 4 novos processos. Após a finalização dos processos, os resultados serão agrupados por uma nova soma (*callback* sum, presente no decorator).

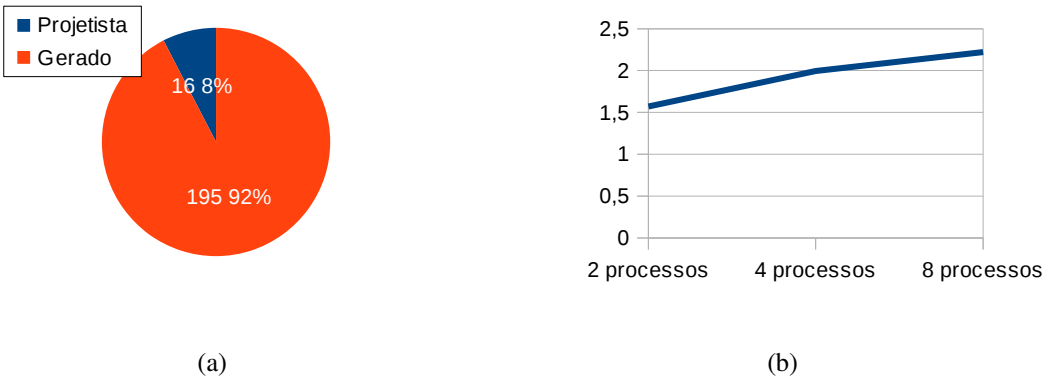
A seção a seguir apresenta alguns dos resultados obtidos até o momento.

3. Resultados Preliminares

Para avaliação da abordagem proposta, adotou-se um estudo de caso que consiste de uma aplicação paralela para regressão linear simples. Para estimar os parâmetros de uma equação linear foi utilizado o Método dos Mínimos Quadrados (MMQ) [WEISBERG, 2005], que dispõe das seguintes equações:

$$a = \frac{n.\sum_{i=1}^{n} x_i.y_i - \sum_{i=1}^{n} x_i.\sum_{i=1}^{n} y_i}{n.\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}, b = \frac{\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i}{n}$$

As equações acima consistem, respectivamente, do intercepto (a) e do coeficiente angular (b) do modelo de regressão linear. Essas fórmulas foram decompostas em função de seus somatórios (Σx, Σy, Σxy e Σx^2), de forma que cada um deles pudesse ser modelado como um vértice de processamento e, consequentemente, ser paralelizado, utilizando a abordagem proposta neste trabalho. Para avaliar o desempenho a aplicação paralela para regressão linear, utilizou-se um conjunto de amostras de consumo de energia elétrica, realizadas de minuto em minuto, durante 47 meses, em uma das tomadas elétricas de uma casa de família. Ao todo, o conjunto possui 2.075.259 registros com 2 variáveis (hora e voltagem). Para o estudo de caso, as variáveis X e Y correspondem ao instante da medição e a voltagem aferida, respectivamente. Os resultados são exibidos nas Figuras 5 (a) e (b).



(a) (b)

Figura 5. Relação entre linhas de código (a) e *Speedup* (b).

Na Figura 5 (a) percebe-se que há uma redução no esforço de programação da aplicação paralela, visto que o número de linhas de código informadas pelo projetista é pequeno (apenas 8%) em relação ao total da aplicação. A Figura 5 (b) apresenta os resultados de desempenho para o estudo de caso, considerando diferentes números de processos para as tarefas paralelas. Para os experimentos utilizou-se um AMD A8 (4 núcleos) com 6GB de RAM. Percebe-se que, em relação à aplicação sequencial, as aplicações paralelas obtiveram maior desempenho, destacando-se o maior *speedup* (2,3x) com 8 processos.

4. Conclusões

Este artigo apresentou uma nova abordagem para suportar a criação visual de novas aplicações paralelas. Propôs-se um novo modelo de grafo para modelar elementos de dados e de processamento (sequencial e paralelo por divisão de domínio) da aplicação. Os resultados mostraram que através da modelagem gráfica, é possível reduzir o esforço para a criação de novas aplicações paralelas de alto desempenho. Trabalhos futuros incluem o suporte: a novas técnicas de paralelismo, a código alvo em outras linguagens de programação e a novos modelos de paralelização (e.g. MPI e CUDA).

Referências

Böhm, S., Běhálek, M., Meca, O., & Šurkovský, M. (2014) Kaira: Development Environment for MPI Applications. In Application and Theory of Petri Nets and Concurrency (pp. 385-394). Springer International Publishing, 2014.

Bouajila, A.; Zeppenfeld, J.; Stechele, W.; Bernauer, A.; Bringmann, O.; Rosenstiel, W. and Herkersdorf, A. (2011) Autonomic System on Chip Platform. In: Organic Computing - A Paradigm Shift for Complex Systems. Springer.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2012) Algoritmos, 3a. Ed. Elsevier.

Gebali, F. (2011) Algorithms and Parallel Computing. John Wiley & Sons.

Gamma, E.; Helm, R.; Johnson, R. and Vlissides, J. (2000) Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos. 5a. Ed. Bookman.

Lee, P. A. and Webber, J. (2003)Taxonomy for Visual Parallel Programming Language. Technical Report Series-University of Newcastle upon Tyne Computing Science.

Mattson, G. T.; Sanders, A. B. and Massingill L. B. (2005) Patterns for Parallel Programming. Addison-Wesley, Boston, MA.

Newton, P. and Browne, J. C. (1992) The CODE 2.0 graphical parallel programming language. In Proc. of the 6th international conference on Supercomputing (pp. 167-177). ACM.

Pacheco, P. S. (2011) An Introduction to Parallel Programming, Morgan Kaufmann Publishers.

Palach, J. (2014) Parallel Programming with Python. Packt Publishing Ltd.

Weisberg, S. (2005) Applied Linear Regression, Willey and Sons, Third Edition.Wiley.