Detecção de Anomalias de Desempenho em Aplicações de Alto Desempenho baseadas em Tarefas em Clusters Híbridos

Vinicius Garcia Pinto[1,2], Lucas Mello Schnorr[1], Arnaud Legrand[2], Samuel Thibault[3], Luka Stanisic[4], Vincent Danjean[2]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul
Porto Alegre, Brasil

[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG,
Grenoble, France

[3]Inria Bordeaux Sud-Ouest
Bordeaux, France

[4]Max Planck Computing and Data Facility
Garching, Germany

{vgpinto, schnorr}@inf.ufrgs.br
{arnaud.legrand, samuel.thibault, vincent.danjean}@inria.fr
luka.stanisic@mpcdf.mpg.de

Abstract. *Programming paradigms in High-Performance Computing have been shifting towards task-based models which are capable to more readily adapt to heterogeneous and scalable supercomputers. Detecting performance anomalies in such environments is particularly difficult since it must consider architecture heterogeneity, variability, and the capability to obtain trusted measurements. This work presents a case-study about the detection of anomalies in the execution of the well-known tiled dense Cholesky factorization developed with StarPU. Our experiments have been conducted in a variety of hybrid multi-node platforms to demonstrate how we are capable to detect and highlight performance anomalies.*

Resumo. *Os paradigmas de programação em Computação de Alto Desempenho estão mudando para modelos baseados em tarefas que são capazes de se adaptar a supercomputadores com arquiteturas heterogêneas e escaláveis. A detecção de anomalias de desempenho em tal cenário é particularmente difícil uma vez que ela deve considerar a heterogeneidade da arquitetura, a variabilidade e a capacidade de obter medições confiáveis. Este trabalho apresenta um estudo de caso sobre a detecção de anomalias na execução da conhecida fatoração de Cholesky por blocos desenvolvida com StarPU. Os experimentos foram conduzidos em uma variedade de plataformas com múltiplos nós híbridos para demonstrar a capacidade de detectar e destacar anomalias de desempenho.*

1. Introdução

As plataformas de Processamento de Alto Desempenho (PAD) têm sido construídas com nós computacionais híbridos visando atender a crescente demanda por poder computacional. Este fato tem provocado uma mudança nos paradigmas de programação para PAD em

direção a modelos baseados em tarefas que executam sobre ambientes de execução, uma vez que estes são capazes de se adaptar mais facilmente a plataformas com arquiteturas heterogêneas e escaláveis. Ambientes de execução automatizam etapas que, tradicionalmente, seriam feitas manualmente pelo programador como o escalonamento das tarefas, balanceamento de carga e as transferências de dados.

Entretanto, uma vez que a execução de aplicações neste modelo de hardware e software é inerentemente estocástica, detectar anomalias de desempenho em tal cenário é particularmente difícil. As ferramentas e técnicas existentes para detecção de anomalias em PAD foram desenvolvidas para análise de aplicações implementadas com o modelo BSP (*bulk-synchronous parallel*). Neste contexto, a identificação de anomalias é relativamente trivial, uma vez que a aplicação é bem estruturada em termos de fases de computação e comunicação e o *hardware* é homogêneo. Por outro lado, em plataformas híbridas e com aplicações baseadas em tarefas, esta detecção é desafiadora pois deve considerar a heterogeneidade da arquitetura, a variabilidade do desempenho e a capacidade de se obter medições confiáveis.

Neste trabalho, nós apresentamos um estudo de caso sobre a detecção de anomalias na execução de uma fatoração de Cholesky implementado com paralelismo de tarefas. Neste contexto, é esperado que a duração das tarefas de um mesmo tipo seja dependente apenas do tipo de recurso de processamento no qual elas são executadas. Assim, uma tarefa anômala é aquela cuja duração é excessivamente longa em comparação com as outras tarefas do mesmo tipo que se executam sobre um recurso de mesma natureza (CPU ou GPU).

Este estudo de caso utiliza nosso *framework* construído sobre ferramentas de análise de dados para visualizar rastros de execução. Nossos experimentos foram executados com o ambiente de execução StarPU [Augonnet et al. 2011] em uma variedade de plataformas híbridas (mesclando CPUs homogêneas e GPUs homogêneas) e os resultados estão divididos em duas contribuições: a detecção de anomalias em núcleos CPU e uma investigação sobre as anomalias na duração das tarefas que executam em GPUs. Embora em trabalhos anteriores [Pinto et al. 2016, Pinto et al. 2018] nós já tenhamos utilizado nosso *framework* para analisar execuções com StarPU, este trabalho foca em um aspecto que não foi discutido previamente. A investigação descrita neste trabalho permite enumerar hipóteses para origem das anomalias identificadas na duração das tarefas assim como a correção destas.

O restante deste artigo está organizado da seguinte forma. A Seção 2 discute aspectos relacionados ao contexto deste trabalho como as arquiteturas híbridas CPU/GPU (2.1), a programação baseada em tarefas (2.2), o ambiente de execução StarPU (2.3) e a fatoração de Cholesky (2.4) bem como os trabalhos relacionados (2.5). A Seção 3 apresenta o nosso *framework* StarVZ (3.1) utilizado para a análise de execuções de aplicações baseadas em tarefas que executam em arquiteturas híbridas com suporte do ambiente de execução StarPU, bem como o modelo utilizado para identificar anomalias na duração das tarefas (3.2). A Seção 4 apresenta os resultados da detecção de anomalias em núcleos CPU (4.2) e em dispositivos GPU (4.3). Por fim, a Seção 5 apresenta as considerações finais. Os dados brutos dos experimentos, o código fonte para processamento dos rastros e geração das figuras estão disponíveis na versão reprodutível deste trabalho em `https://gitlab.in2p3.fr/vgarciap/wperformance2018`.

2. Contexto e Trabalhos Relacionados

2.1. Arquiteturas Híbridas CPU/GPU

A estagnação da frequência dos processadores tem incentivado a adoção de outros meios para atender a demanda de desempenho das aplicações atuais. Um destes meios tem sido a construção de arquiteturas híbridas, combinando recursos de processamento distintos em um mesmo nó. Muitas plataformas de computação de alto desempenho atuais são constituídas de nós computacionais híbridos, onde vários dispositivos aceleradores são acoplados a processadores tradicionais para aumentar a capacidade computacional do sistema. Esta tendência pode ser confirmada por meio da análise da edição de Novembro de 2017 da lista Top500, onde 102 dentre os 500 sistemas classificados utilizam algum tipo de dispositivo acelerador como GPUs e Xeon Phi [Meuer et al. 2014].

2.2. Programação Paralela baseada em Tarefas

A programação baseada em tarefas tem sido adotada como uma alternativa para tratar a heterogeneidade das plataformas atuais de computação de alto desempenho. Tradicionalmente, as aplicações de alto desempenho são implementadas seguindo o modelo BSP utilizando ferramentas clássicas e explícitas como MPI, OpenMP, *pthreads* e CUDA. Entretanto, em plataformas híbridas, este modelo leva a códigos monolíticos, que são difíceis de desenvolver, otimizar e manter, embora ainda possibilite que se atinja um desempenho próximo do ideal.

Por outro lado, o modelo baseado em tarefas se baseia em ambientes de execução dinâmicos visando abstrair a complexidade da plataforma e explorar de maneira eficiente os múltiplos níveis de paralelismo do *hardware*. Neste modelo, o programador descreve a aplicação em termos de tarefas abstratas e suas dependências. O ambiente de execução é responsável por atividades relacionadas à plataforma como escalonamento das tarefas, transferências de dados e gerenciamento das dependências. Para executar estas atividades, o ambiente de execução analisa características da plataforma e do código, tais como as implementações fornecidas para uma tarefa, os recursos de processamento disponíveis (CPUs, GPUs, etc), a duração estimada das tarefas, a localidade dos dados e a largura de banda da interconexão. A partir das informações coletadas, o ambiente de execução pode usar heurísticas de escalonamento adequadas e implementar otimizações, como por exemplo, antecipar as transferências de dados, visando atingir melhor desempenho. Como exemplo de ferramentas com suporte ao modelo de tarefas em arquiteturas híbridas, pode-se citar: OmpSs [DURAN et al. 2011], XKaapi [Gautier et al. 2013] e StarPU [Augonnet et al. 2011].

2.3. StarPU

O StarPU [Augonnet et al. 2011] é um ambiente de execução para programação baseada em tarefas em arquiteturas híbridas. O ambiente foi inicialmente projetado para gerenciar plataformas híbridas compostas por um único nó de processamento equipado com processadores *multicore* (CPUs) e aceleradores (GPUs, Intel Xeon Phi). De forma a explorar de maneira eficiente o paralelismo da plataforma, o StarPU utiliza múltiplas implementações para uma mesma tarefa, por exemplo, com versões equivalentes para CPUs e/ou GPUs.

O escalonador utilizado pelo StarPU decide em tempo de execução onde executar as tarefas levando em conta os recursos de processamento disponíveis, o tipo destes, a localização atual dos dados a serem acessados pela tarefa e as implementações disponíveis

para a mesma. Este escalonador oferece várias políticas de escalonamento. As políticas DMDA e DMDAS têm base em modelos de desempenho pré-calibrados, enquanto as políticas WS e LWS utilizam uma estratégia baseada em roubo de tarefas, roubando tarefas do recurso mais carregado (WS) ou do vizinho mais carregado (versão LWS). A política PRIO se baseia apenas em anotações de prioridade especificadas pelo programador da aplicação.

Os algoritmos DMDA (*Deque Model Data Aware*) e DMDAS (*Deque Model Data Aware Sorted*) são membros de uma família de escalonadores do StarPU que utilizam a duração prevista da tarefa e a duração prevista das transferências de dados para executar o escalonamento das tarefas. Ambas estratégias têm base na técnica de escalonamento de listas (*list scheduling*). Os algoritmos DMDA e DMDAS são heurísticas gulosas que escalonam tarefas na ordem em que elas se tornam disponíveis. Quando da tomada das decisões de escalonamento, estas técnicas levam em conta o tempo previsto para execução da tarefa, a duração estimada das transferências de dados entre CPUs e GPUs e o desempenho relativo dos recursos de processamento para cada *kernel* computacional. O algoritmo DMDAS melhora as decisões do DMDA por meio da ordenação das tarefas em listas em cada recurso, utilizando como critério de ordenação o número de fatias de dados já transferidas e a prioridade, o que pode ser custoso quando o número de tarefas é elevado. Este algoritmo, é portanto, mais próximo da proposta original do algoritmo HEFT (*Heterogeneous Earliest Finish Time*) que respeita as prioridades e leva em conta as decisões passadas de escalonamento.

Os algoritmos WS (*Work Stealing*) e LWS (*Locality Work Stealing*) utilizam uma lista de tarefas por recurso de processamento; novas tarefas são armazenadas localmente por padrão. Quando um recurso se torna ocioso, na política WS, ele rouba tarefas do recurso mais sobrecarregado. A política LWS, por outro lado, impõe que o recurso ocioso deve, em um primeiro momento, tentar roubar do recurso mais sobrecarregado dentre os seus vizinhos. Esta escolha da vítima difere do algoritmo clássico de roubo de tarefas [Blumofe and Leiserson 1999] onde a escolha da vítima é feita por meio de uma estratégia aleatória.

O algoritmo PRIO utiliza uma simples lista centralizada que é compartilhada entre todos os recursos de processamento. Esta lista mantém as tarefas ordenadas respeitando as prioridades especificadas pelo programador da aplicação.

A extensão StarPU-MPI [Augonnet et al. 2012, Agullo et al. 2017] fornece suporte adicional para gerenciar arquiteturas com múltiplos nós, adotando a especificação MPI [Gropp et al. 1999] como meio de acesso à rede. A principal característica desta extensão é que ela utiliza um escalonador StarPU independente para cada nó. Por exemplo, se um experimento é configurado com a política DMDA, este escalonador será usado em cada um dos nós para processar uma parte do grafo acíclico direcionado (DAG, abreviação em inglês de *Directed Acyclic Graph*) da aplicação. As dependências existentes entre os nós são resolvidas através de operações MPI *send*-/receive/ ponto-a-ponto, e gerenciadas como qualquer outra tarefa escalonada pelo ambiente de execução. Uma vez que a decomposição de domínio é estática, estas operações *send*-/receive/ são submetidas automaticamente como tarefas do DAG. StarPU-MPI utiliza uma *thread* especificamente para gerenciar as comunicações MPI do nó. Ao concluir uma tarefa cuja saída é requerida por uma tarefa em outro nó MPI, a *thread* MPI submete a operação *send* correspondente. No

outro nó, quando da recepção dos dados por MPI, a *thread* MPI habilita a execução da tarefa correspondente.

A distribuição bidimensional clássica *Block-Cyclic Distribution* [Blackford et al. 1997] foi previamente modificada para suportar execuções com decomposição estática em múltiplos nós com apoio do StarPU-MPI. A decomposição depende dos parâmetros P × Q e do número de nós MPI. Em função do numero de nós MPI, os parâmetros P e Q controlam como a matriz de entrada é particionada entre os nós com base nos blocos. O valor de P pode variar de um até o número de nós. Tomando como exemplo uma matriz de 16×16 blocos, uma configuração de P=1 × Q=8 leva a uma distribuição por linhas, enquanto uma configuração de P=8 × Q=1 leva a uma distribuição por colunas. Combinações de valores intermediários levam a distribuições intercaladas. Em um cenário ideal, para um dado número de nós, o valor de P deveria ser definido de forma a minimizar o perímetro de comunicação de cada nó uma vez que ele é relativo ao volume total de comunicações.

2.4. Cholesky

Neste trabalho, utilizamos a fatoração de matrizes pelo método de Cholesky como aplicação de estudo de caso. Este método é uma das operações mais comuns de álgebra linear e é usado por várias aplicações científicas. Com objetivo de melhorar a reprodutibilidade e a estabilidade de nossos experimentos, nós adotamos a implementação da fatoração de Cholesky por blocos (*tiled*) fornecida pelo pacote de álgebra linear Chameleon/MORSE [Agullo et al. 2012]. Esta implementação é construída sobre o ambiente de execução StarPU [Augonnet et al. 2011] e compilada com bibliotecas BLAS (CPU) e CUBLAS (GPUs) padrão.

A Figura 1 ilustra uma versão simplificada desta aplicação. As linhas com as chamadas `dpotrf`, `dtrsm`, `dsyrk` e `dgemm` representam a criação de tarefas StarPU com implementações de precisão dupla para CPUs e GPUs (exceto o `dpotrf` que possui apenas a versão CPU). Os parâmetros destacados RW e R indicam o modo de acesso do bloco da matriz em questão. A partir destas informações sobre os modos de acesso, o ambiente de execução pode inferir as dependências entre as tarefas e então construir um grafo direcionado acíclico (DAG) de tarefas. A cada iteração k do laço mais externo, uma tarefa `dpotrf` habilita a execução de $N - k - 1$ `dtrsm`, e então $N - k - 1$ `dsyrk`, sucedida por $\approx (N-k)^2/2$ tarefas `dgemm`. A partir das dependências, podemos observar que várias iterações podem ser executadas simultaneamente e que o número de repetições nos laços mais internos diminui quando k aumenta. Por fim, o tempo de execução de uma tarefa depende altamente do seu tipo (`dpotrf`, `dtrsm`, `dsyrk` e `dgemm`) e do tipo de recurso onde ela é executada (CPU ou GPU).

Embora neste trabalho nós tenhamos utilizado a fatoração de Cholesky como estudo de caso, existem outras aplicações que apresentam características similares e que portanto poderiam ser analisadas com os mesmos métodos e ferramentas aqui descritos. Podemos citar os algoritmos por blocos para fatoração de matrizes LU e QR que, assim como o método de Cholesky, utilizam como base chamadas à operações do padrão BLAS (*Basic Linear Algebra Subprograms*).

```
for (k = 0; k < N; k++) {
  DPOTRF(RW, A[k][k]);
  for (i = k+1; i < N; i++)
    DTRSM(RW, A[i][k], R, A[k][k]);
  for (i = k+1; i < N; i++) {
    DSYRK(RW, A[i][i], R, A[i][k]);
    for (j = k+1; j < i; j++)
      DGEMM(RW, A[i][j], R, A[i][k], R, A[j][k]);
  }
}
```

Figura 1. Pseudo-código da decomposição de Cholesky por blocos

2.5. Trabalhos Relacionados

Existem poucas ferramentas para análise de desempenho de aplicações baseadas em tarefas. Entre as ferramentas que não possuem foco em arquiteturas híbridas, podemos citar o Framesoc/Ocelotl [Dosimont et al. 2015], que oferece técnicas escaláveis para agregação de rastros. Estas técnicas agregam comportamentos homogêneos apresentados tanto na dimensão espacial (recursos de processamento) quando temporal (estados da aplicação). Comportamentos não homogêneos como anomalias são mantidos em seu estado bruto e são visualizados sem agregação. O controle sobre quais dados são considerados homogêneos ou não é feito por uma heurística baseada na relação entre a perda de informação e a complexidade da visualização. Dessa forma, a detecção de anomalias globais, como aquelas que ocorrem em todas as tarefas de um core, pode ser prejudicada, uma vez que a ferramenta tende a identificar este comportamento como homogêneo.

No contexto de aplicações com tarefas em arquiteturas híbridas, nós apresentamos previamente uma discussão preliminar dos experimentos aqui apresentados [Schnorr et al. 2017]. Uma análise similar, utilizando a mesma aplicação (cholesky do pacote Chameleon) e o mesmo ambiente de execução (StarPU) foi feita por [Beaumont et al. 2018]. Neste estudo, o objetivo dos autores foi avaliar o impacto da presença de tarefas com duração atípica no tempo total de execução da aplicação (*makespan*). A análise foi conduzida com o escalonador DMDAS em uma plataforma single-node com 24 cores e 4 GPUs. Os autores sugerem que as anomalias podem estar relacionadas a efeitos da arquitetura NUMA (*Non Uniform Memory Access*) da plataforma. Dessa forma a análise das anomalias não foi feita individualmente, mas por grupos de *cores* (correspondentes aos nós NUMA). Por fim, para avaliar o impacto da presença de tarefas anômalas no *makespan*, os autores fizeram uso de execuções simuladas utilizando StarPU-Simgrid [Stanisic et al. 2015] com injeção de tempos anômalos para algumas tarefas conforme as hipóteses enumeradas. Os principais diferenciais de nosso trabalho em relação aos demais aqui discutidos são: (a) o emprego de uma análise visual dos rastros de execução da aplicação nas dimensões espacial (recursos de processamento) e temporal (duração das tarefas), (b) nossa análise considera execuções em arquiteturas distribuídas com múltiplos nós híbridos, (c) nosso estudo de caso inclui testes com outros escalonadores além do DMDAS (padrão) e (d) nosso estudo permite enumerar hipóteses para origem e correção das causas das anomalias.

3. *Framework* para análise de aplicações com tarefas e Modelo de desempenho para tarefas de álgebra linear densa

Nesta seção, apresentamos o *framework* utilizado para analisar execuções de aplicações baseadas em tarefas que executam em arquiteturas híbridas.

3.1. StarVZ

Em trabalhos anteriores [Pinto et al. 2016, Pinto et al. 2018] nós propusemos um *framework*[1] baseado em uma estratégia modular e incremental que permite representar execuções de maneira visual utilizando rastros de execução previamente coletados. Esta solução é construída sobre ferramentas modernas para análise de dados, combinando *pj_dump* [Schnorr et al. 2016], a linguagem de programação R [R Core Team 2017], incluindo sua biblioteca *ggplot2* [Wickham 2009] e as funções para manipulação de dados fornecidas pelo meta-pacote *tidyverse* [Wickham 2016], além de org-mode [Schulte et al. 2012] e *plotly* [Sievert et al. 2016].

Este *framework* permite obter, a partir dos rastros de execução, um diagrama espaço/tempo representando a duração de cada uma das tarefas da aplicação nos recursos de processamento ao longo do tempo. Graças a estratégia modular e incremental, podemos enriquecer esta visualização básica adicionando informações relevantes como arestas de dependências entre as tarefas, estimativas para o *makespan*, porcentagem de tempo ocioso e tarefas com duração anômala. Além disso, por meio de gráficos auxiliares, podemos adicionar outras informações relevantes como a quantidade de tarefas disponível para execução em cada instante, a repartição de trabalho entre os recursos de processamento de diferentes tipos e a progressão da computação ao longo do tempo.

Em trabalhos anteriores mostramos como este conjunto de técnicas pode ajudar a identificar e corrigir problemas de escalonamento em nós híbridos equipados com CPUs *multicore* e múltiplas GPUs [Pinto et al. 2016] e como identificar e corrigir gargalos em execuções com nós híbridos em plataformas distribuídas [Pinto et al. 2018]. Neste estudo, focamos na detecção de anomalias de desempenho na duração das tarefas, buscando identificar a origem e possíveis soluções para as mesmas.

3.2. Modelo de desempenho para detecção de anomalias na duração das tarefas

No contexto do tipo de aplicação considerado neste trabalho, uma operação de álgebra linear densa implementada utilizando o modelo de programação paralela baseado em tarefas, é esperado que a duração das tarefas de um mesmo tipo seja dependente apenas do tipo de recurso de processamento no qual elas são executadas. Esta restrição é particularmente relevante no caso de aplicações já bastante otimizadas, como a fatoração de matrizes pelo método de Cholesky apresentada na Seção 2.4. Nesta aplicação, as tarefas executam chamadas a operações do padrão BLAS, que são comuns a muitas outras aplicações como as fatorações LU e QR.

Neste trabalho, nós utilizamos a seguinte Equação (1) para identificar as tarefas com duração anômala. A duração de uma tarefa t de um dado tipo quando executada em um certo recurso de processamento (core ou GPU) é considerada anormal se a sua duração t_d:

[1] `https://github.com/schnorr/starvz`

$$t_d \geq Q_3 + 1.5 \times IQR \quad (1)$$

onde Q_3 é o terceiro quartil e IQR é a diferença ($Q_3 - Q_1$), sendo que Q_1 é o primeiro quartil. Os quartis são computados considerando globalmente todas as tarefas de um mesmo tipo, e que executam em um recurso de mesma natureza (core CPU ou GPU). Posteriormente, nós aplicamos esta Equação para marcar as tarefas com duração anômala.

O uso desta Equação, tipicamente usado para definir *outliers* em *boxplots*, é debatível e foi baseado em nossa experiência. De qualquer maneira, em nossa abordagem, esta fórmula é facilmente substituível. Entretanto, ela atende aos requisitos do nosso estudo de caso uma vez que considera tanto a heterogeneidade da aplicação (tipo de tarefa) quanto dos recursos de processamento (GPU, CPU).

4. Avaliação Experimental e Resultados

Nesta seção apresentamos a avaliação experimental de nossa abordagem para detecção de anomalias. Os experimentos estão divididos em duas partes: a primeira apresenta a análise do desempenho da aplicação fatoração de Cholesky quando executada em um *cluster* de nós híbridos enquanto a segunda apresenta a análise de *microbenchmarks* com chamadas diretas ao *kernel* `dgemm` da biblioteca cuBLAS[2], executados diretamente em uma GPU sem o uso do ambiente StarPU. As execuções da fatoração de Cholesky foram feitas com matrizes de dois tamanhos: 108×108 blocos de 1440×1440 e 35×35 blocos de 960×960.

4.1. Ambiente Experimental

Os experimentos aqui discutidos foram executados em arquiteturas híbridas compostas de CPUs *multicore* e de GPUs conforme detalhado na Tabela 1. A plataforma chifflet foi configurada com Debian Linux (kernel 4.9.0-3-amd64), gcc 6.3.0, Chameleon 0.9.1 (master), StarPU (developer, com revisões específicas para as correções propostas neste trabalho, OpenMPI 2.0.2, OpenBLAS 0.2.19 e CUDA 7.0 com *driver* 375.66. A plataforma Hype5, onde foram executados os *microbenchmarks*, foi configurada com Ubuntu Linux (kernel 4.4.0-112-generic), gcc 5.4.1 e CUDA 9.0 com driver 384.111.

Tabela 1. Plataformas Experimentais Utilizadas

Nome	Nós	Processador do Nó	GPUs	Rede
chifflet	8	2 × 14-core Intel Xeon E5-2680v4	2 × Nvidia GTX 1080Ti	Ethernet 10Gbit
Tupi	1	1 × 8-core Intel Xeon E5-2620v4	2 × Nvidia GTX 1080Ti	(não aplicável)
Hype5	1	2 × 10-core Intel Xeon E5-2650v3	2 × Nvidia Tesla K80	(não aplicável)

[2] `http://docs.nvidia.com/cuda/cublas/index.html`

4.2. Anomalias em Núcleos CPU

A Figura 2 apresenta um cenário representativo da visualização de rastros utilizando a detecção de anomalias discutida anteriormente. A análise desta execução permite identificar um comportamento instigante uma vez que a presença de tarefas com duração anormal é mais concentrada em alguns *cores* que em outros. Tarefas anomalamente longas ocorrem majoritariamente no *core* 0 de cada nó da plataforma, o que pode ser verificado nos gráficos da Figura 3. Devido ao fato que o problema aparece independente da quantidade de cores utilizada no experimento, preferimos ilustrar o problema de forma mais concisa através de um experimento com apenas 6 cores e 2 GPUs em cada um dos oito nós.

Figura 2. Anomalias (marcadas por cor mais forte) em um *core* (principalmente CPU0) em cada nó da plataforma *chifflet* durante a execução da fatoração de Cholesky em uma matriz com 108×108 blocos de 1440×1440 utilizando o escalonador LWS e P=4.

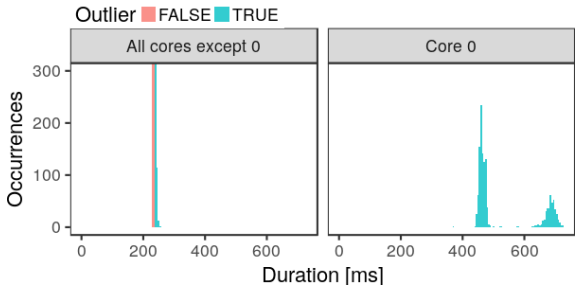

Figura 3. Comparação da ocorrência de anomalias em todos os cores exceto no Core 0 contra o cenário que considera apenas o Core 0. Em vermelho, a ocorrência de tarefas cuja duração é considerada normal pelo nosso modelo. Em azul, a ocorrência de tarefas cuja duração é avaliada como anômala.

Por padrão as *workers threads* do StarPU são fixadas aos *cores*, ou seja, *threads* que executam computação diretamente e aquelas que controlam as GPUs são associadas aos cores em uma distribuição 1 para 1 e a migração durante a execução não é permitida. Entretanto, além das *workers threads* existem duas outras *threads* que não são fixadas aos *cores*. A primeira é a thread principal responsável pela criação e submissão das tarefas enquanto a segunda gerencia as comunicações MPI.

A hipótese inicial para explicar as anomalias na duração das tarefas é de que as *threads* adicionais (submissão de tarefas e MPI) estejam executando no *core* 0, perturbando a *worker* thread fixada neste core. Para verificar esta hipótese, foram implementadas modificações no StarPU de forma que o usuário possa, manualmente, fixar cada uma das *threads* (*worker*, MPI e principal) a um *core* específico.

A Figura 4 apresenta uma execução em dois nós (marcados no eixo vertical por "0_"e "1_", cada um com 5 cores e 2 GPUs) utilizando dois escalonadores diferentes: DMDAS no painel da esquerda e LWS na direita. O experimento tem por objetivo verificar a efetividade da solução proposta. Podemos notar que as anomalias não estão mais presentes no core 0 de cada nó. Entretanto, ainda existem *cores* individualmente afetados como o core 2 do nó 0 da execução com DMDAS e o core 3 do nó 1 na execução com LWS. Experimentos adicionais indicam que as anomalias estão ou localizadas espacialmente (afetando um *core*) ou ausentes (sem correlação significativa com os *cores*).

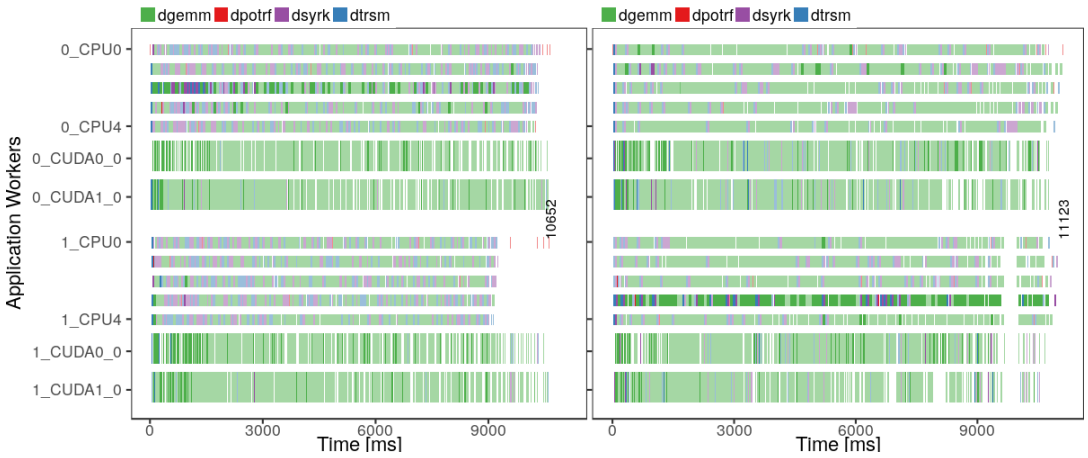

Figura 4. Execuções da fatoração de Cholesky em dois nós após as modificações no StarPU. Matriz com 35×35 blocos de 960×960 utilizando os escalonadores DMDAS (painel da esquerda) e LWS (direita) e P=1.

Uma investigação mais aprofundada, inspecionando a execução através do depurador GDB indica que o problema pode estar relacionado à implementação da biblioteca cuBLAS, cujo código é proprietário, ou às configurações de afinidade definidas pelo sistema operacional na inicialização da aplicação.

4.3. Anomalias em GPUs

As anomalias discutidas na Seção 4.2 ocorrem principalmente em *workers* executando em CPUs. Entretanto, também podemos observar na Figura 2 que existem anomalias nos *workers* que executam em GPUs. Neste caso, o comportamento parece ser comum a todas as GPUs, sem indicação de que exista uma correlação entre as anomalias e alguma GPU

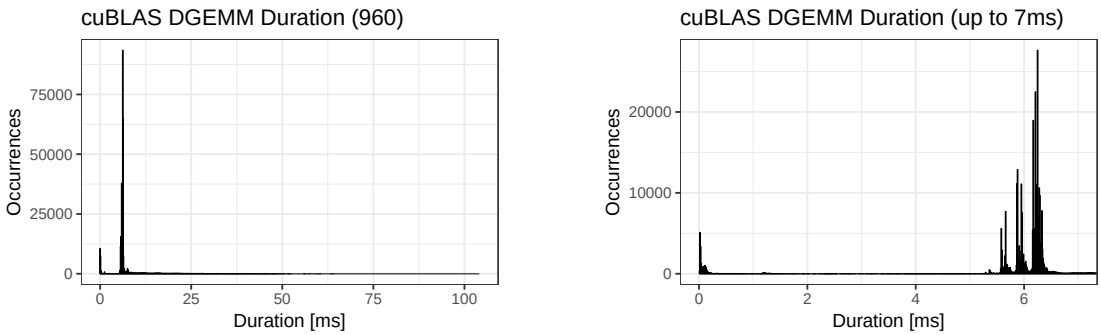

Figura 5. Histograma da duração das tarefas `dgemm` que executam em GPUs com matrizes de 960×960. O gráfico da direita apresenta um *zoom* nas tarefas com duração de até 7ms.

específica. Uma vez que as GPUs executam somente computação, ao contrário das CPUs do caso anterior, elas não são passíveis de perturbação por *threads* de gerenciamento.

Uma hipótese para explicar a origem das tarefas com duração anômala em GPUs pode ser a imprecisão do modelo de desempenho utilizado. É possível que este modelo esteja considerando como atípicas, tarefas que estariam dentro da variabilidade esperada para execução neste *hardware*. A Figura 5 apresenta o histograma da duração das tarefas `dgemm` que executam em GPUs. A análise deste histograma mostra que a distribuição da duração destas tarefas é multimodal. O que pode ser a razão do número elevado de tarefas cuja duração é identificada como anômala pelo nosso modelo.

Uma possível explicação para a distribuição multimodal pode estar relacionada a efeitos de *warm-up* e de *cache*. A Figura 6 apresenta os resultados de um conjunto de experimentos para investigar estas hipóteses. Nestes experimentos, nós executamos diretamente chamadas ao *kernel* `dgemm` da biblioteca cuBLAS, variando o intervalo entre as execuções entre 1 e 20ms (colunas) bem como o tamanho da matriz de entrada (linhas) entre 128×128 e 1960×1960. Pode-se observar que para que se obtenha benefício do *warm-up*, o tempo entre duas execuções não deve ser maior que a duração total do *kernel* em execução. Isto provavelmente pode estar relacionado a uma sequência de execuções do mesmo *kernel*, o que pode explicar a identificação de anomalias nas execuções em *workers* GPU com o StarPU. Além disso, quanto maior a duração do *kernel* maiores as oportunidades de se observar os resultados do *warm-up*, e a consequente aceleração.

5. Considerações Finais

Neste artigo nós apresentamos um estudo de caso sobre a detecção de anomalias na duração das tarefas de uma aplicação que executa sobre nós híbridos. A detecção deste tipo de irregularidade é desafiadora dadas as características da arquitetura híbrida e das técnicas implementadas pelo ambiente de execução. O estudo foi conduzido com o nosso *framework* baseado na linguagem de programação R e que possibilita a análise de rastros de execução obtidos com o ambiente de execução StarPU e plataformas híbridas multi-(nó, *core*, GPU).

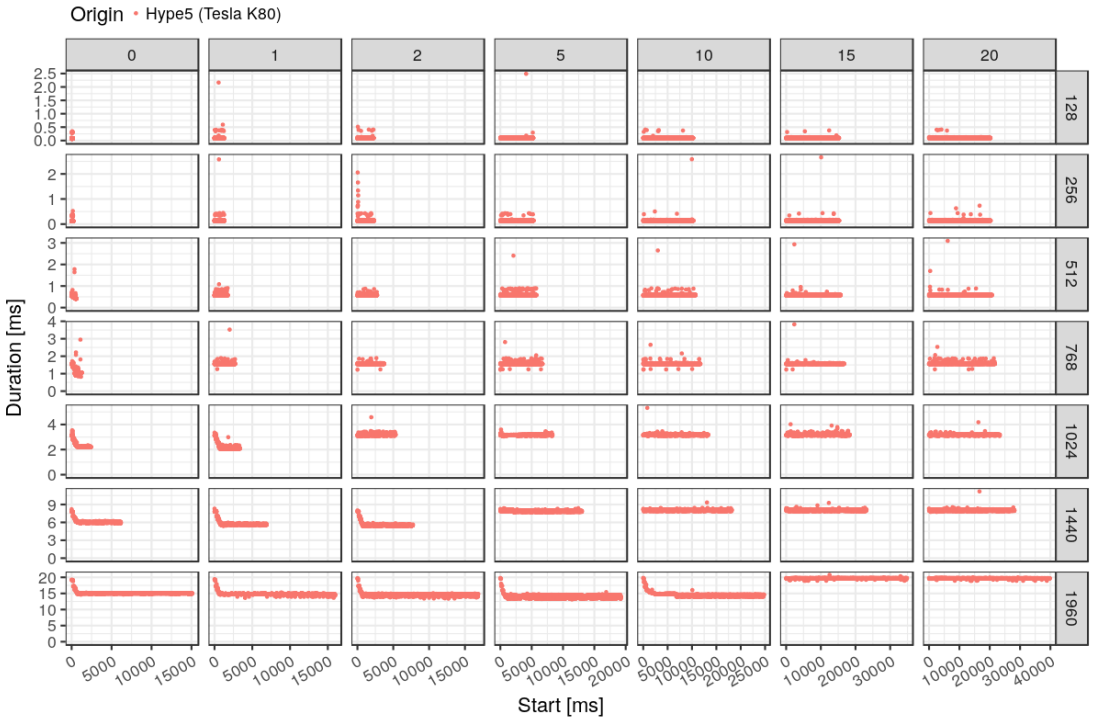

Figura 6. Análise da execução do *kernel* `dgemm` do cuBLAS. As colunas indicam o intervalo em mili-segundos entre duas execuções, enquanto as linhas indicam o tamanho da matriz de entrada.

Por meio dos resultados obtidos pela análise da fatoração de Cholesky fornecida pelo pacote Chameleon foi possível identificar e caracterizar anomalias na duração tanto de tarefas que executam em núcleos CPU quanto em GPUs. Em trabalhos futuros nós pretendemos estender esta análise utilizando aplicações de álgebra linear esparsa e também investigar execuções em *clusters* com processadores *many-core* como o Intel Xeon Phi.

Agradecimentos

Nós agradecemos os projetos CAPES/Cofecub 764-13, FAPERGS/Inria ExaSE, FAPERGS 16/2551-0000 354-8 (PPP 2014), FAPERGS 16/2551-0000 488-9 (PRONEX 2014), CNPq 447311/2014-0, o Laboratório Internacional CNRS/LICIA, o programa H2020 da União Europeia e o MCTI/RNP através do projeto HPC4E, identificador 689772. Alguns experimentos foram executados na plataforma Grid'5000 (`https://www.grid5000.fr`), com suporte do Inria, CNRS, RENATER e várias outras organizações francesas. O material auxiliar deste artigo[3], incluindo detalhes experimentais, código fonte e trechos de código para regerar as figuras é graciosamente mantido pelo CNRS-IN2P3 (`http://www.in2p3.fr/`), para o qual nós também enviamos nossos agradecimentos.

Referências

[Agullo et al. 2017] Agullo, E., Aumage, O., Faverge, M., Furmento, N., Pruvost, F., Sergent, M., and Thibault, S. P. (2017). Achieving high performance on supercomputers

[3] `https://gitlab.in2p3.fr/vgarciap/wperformance2018`

with a sequential task-based programming model. *IEEE Transactions on Parallel and Distributed Systems*, Early Access:1–14.

[Agullo et al. 2012] Agullo, E., Bosilca, G., Bramas, B., Castagnede, C., Coulaud, O., Darve, E., Dongarra, J., Faverge, M., Furmento, N., Giraud, L., Lacoste, X., Langou, J., Ltaief, H., Messner, M., Namyst, R., Ramet, P., Takahashi, T., Thibault, S., Tomov, S., and Yamazaki, I. (2012). Poster: Matrices over runtime systems at exascale. In Wasserman, H., editor, *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, pages 1332–1332.

[Augonnet et al. 2012] Augonnet, C., Aumage, O., Furmento, N., Namyst, R., and Thibault, S. (2012). StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. In Träff, J. L., Benkner, S., and Dongarra, J. J., editors, *Recent Advances in the Message Passing Interface: 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings*, pages 298–299. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Augonnet et al. 2011] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. (2011). StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience*, 23(2):187–198.

[Beaumont et al. 2018] Beaumont, O., Eyraud-Dubois, L., and Gao, Y. (2018). Influence of Tasks Duration Variability on Task-Based Runtime Schedulers. Research report, INRIA.

[Blackford et al. 1997] Blackford, L. S., Choi, J., Cleary, A., D'Azeuedo, E., Demmel, J., Dhillon, I., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., and Whaley, R. C. (1997). *ScaLAPACK user's guide*. Society for Industrial and Applied Mathematics.

[Blumofe and Leiserson 1999] Blumofe, R. D. and Leiserson, C. E. (1999). Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748.

[Dosimont et al. 2015] Dosimont, D., Corre, Y., Schnorr, L. M., Huard, G., and Vincent, J.-M. (2015). Ocelotl: Large Trace Overviews Based on Multidimensional Data Aggregation. In Niethammer, C., Gracia, J., Knüpfer, A., Resch, M. M., and Nagel, W. E., editors, *Tools for High Performance Computing 2014*, pages 137–160. Springer International Publishing.

[DURAN et al. 2011] DURAN, A., AYGUADÉ, E., BADIA, R. M., LABARTA, J., MARTINELL, L., MARTORELL, X., and PLANAS, J. (2011). Ompss: A proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173–193.

[Gautier et al. 2013] Gautier, T., Lima, J. V., Maillard, N., and Raffin, B. (2013). Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1299–1308. IEEE.

[Gropp et al. 1999] Gropp, W., Thakur, R., and Lusk, E. (1999). *Using MPI-2: Advanced features of the message passing interface*. MIT press.

[Meuer et al. 2014] Meuer, H. W., Strohmaier, E., Dongarra, J., and Simon, H. D. (2014). *The TOP500: History, Trends, and Future Directions in High Performance Computing*. Chapman & Hall/CRC, 1st edition.

[Pinto et al. 2018] Pinto, V. G., Schnorr, L. M., Stanisic, L., Legrand, A., Thibault, S., and Danjean, V. (2018). A visual performance analysis framework for task-based parallel applications running on hybrid clusters. *Concurrency and Computation: Practice and Experience*, Early Access:1–27.

[Pinto et al. 2016] Pinto, V. G., Stanisic, L., Legrand, A., Schnorr, L. M., Thibault, S., and Danjean, V. (2016). Analyzing dynamic task-based applications on hybrid platforms: An agile scripting approach. In *Third Workshop on Visual Performance Analysis, VPA@SC 2016, Salt Lake, UT, USA, November 18, 2016*, pages 17–24.

[R Core Team 2017] R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Schnorr et al. 2016] Schnorr, L. M., Faverge, M., Trahay, F., de Oliveira Stein, B., and de Kergommeaux, J. C. (2016). The Paje trace file format. Technical report, UFRGS.

[Schnorr et al. 2017] Schnorr, L. M., Legrand, A., Thibault, S., Stanisic, L., Pinto, V. G., and Danjean, V. (2017). Detecting performance outliers for task-based hpc applications in multi-[cpu|gpu|node] clusters. Presentation at Workshop on Hybrid Computing 2017, Held in conjunction with SBAC-PAD 2017.

[Schulte et al. 2012] Schulte, E., Davison, D., Dye, T., Dominik, C., et al. (2012). A multi-language computing environment for literate programming and reproducible research. *Journal of Statistical Software*, 46(3):1–24.

[Sievert et al. 2016] Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., and Despouy, P. (2016). *plotly: Create Interactive Web Graphics via 'plotly.js'*. R package version 4.5.6.

[Stanisic et al. 2015] Stanisic, L., Agullo, E., Buttari, A., Guermouche, A., Legrand, A., Lopez, F., and Videau, B. (2015). Fast and Accurate Simulation of Multithreaded Sparse Linear Algebra Solvers. In *The 21st IEEE International Conference on Parallel and Distributed Systems*, pages 481–490, Melbourne, Australia.

[Wickham 2009] Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.

[Wickham 2016] Wickham, H. (2016). *tidyverse: Easily Install and Load 'Tidyverse' Packages*. R package version 1.0.0.