Ferramentas de Análise de Desempenho e Energia: Visão geral, instalação e uso

Gabrieli D. Silva
Vinícius P. Klôh
André Yokoyama
Mariza Ferro
Bruno Schulze
LNCC – Petrópolis,RJ

Abstract

Scientific computing generally requires huge processing power resources' to perform large scale experiments and simulations in a reasonable time. These demands have been addressed by High Performance Computing (HPC), which makes them invaluable tool for modern scientific research. Nowadays, the simulations performed on the petascale supercomputers can achieve accuracy and fidelity never seen before. Despite this, several areas require more computational power to perform their simulations. But, achieving exaflop capabilities imposes some challenging tasks, such as the performance evaluation in HPC, the understanding of the computational requirements of scientific applications, its relation with power consumption, and ways to monitor a wide range of parameters in heterogeneous environments.

To this end, we were finding solutions to enable the monitoring of these relevant parameters for analysis of scientific applications in HPC environments. This is not a trivial task, because it imposes to measure a set of distinct parameters and there isn't an unique monitoring tool able to capture all these aspects and provide them for analysis. For this reason we investigated a range of monitoring tools to enable a high-resolution profile of performance and power consumption with low overhead.

Throughout this study it was noted other difficulties, which can greatly increase the time to the development of research, such as: i) each tool has different approches to measure performance (online, offline, sampling, timing, counting and tracking) and power; ii) different dependencies and complexities for the installation process; iii) works on only for some operational system or programmm language and iv) collect a different set of parameters;

Therefore, this report presents different performance and power monitoring tools, addressing, for each of them, the techniques used for data collection, the structure of the tool for its operation and how the data obtained are presented to the user for analysis. In addition, the details for the installation and the use of the tools are also presented. The main contribution of this report is to be a tutorial for how to install and use, as this process is often costly, since most tools have many dependencies and configurations. Despite these, we present our experience when using the tools and for some of them pointing out its strengths and limitations.

Junho 2018

Sumário

1 Introdução

A Computação de Alto Desempenho (HPC) tem sido amplamente utilizada por oferecer uma infraestrutura computacional que permite a resolução de problemas complexos e que exigem grande poder computacional. Apesar do elevado desempenho dos atuais supercomputadores petaflópicos[1], algumas áreas necessitam de maior poder computacional, pois suas simulações numéricas ainda não podem ser realizadas ou não alcançam a precisão desejada, ou seja, seus desafios científicos ainda são complexos demais para os recursos computacionais disponíveis. A fim de atender essa demanda pretende-se chegar a nova geração de supercomputadores que atinjam a exaescala[2] de processamento. Porém, para que essa nova geração seja economicamente viável, é necessário reduzir o consumo energético desses ambientes.

Para alcançar essa nova geração, monitorar aplicações, ambientes, degradação de desempenho e consumo energético são tarefas indispensáveis. Por meio do monitoramento e da avaliação de desempenho das aplicações científicas é possível caracterizar, para os diferentes modelos e tamanhos de problema, quais seus principais requisitos computacionais, como por exemplo CPU, E/S e memória, e qual a relação desses requisitos com o consumo de energia. Esse tipo de análise viabiliza encontrar soluções para a computação científica, tais como a melhoria do desempenho e a redução do consumo energético. Para essa análise é fundamental a utilização de ferramentas de monitoramento de desempenho e energia. Porém, utilizar essas ferramentas não é tarefa trivial, pois elas possuem diferentes técnicas para o registro de informações comportamentais (amostragem, cronometragem, contagem e rastreamento), as quais implicam diretamente no *overhead* causado pela coleta. Além disso, cada ferramenta obtêm um conjunto de parâmetros diferentes e possui suas particularidades quanto à instalação e uso.

Portanto, este relatório apresenta diferentes ferramentas de monitoramento de desempenho e energia, abordando, para cada uma delas, as técnicas utilizadas para a coleta de dados, a estrutura da ferramenta para o seu funcionamento e como os dados obtidos são apresentados ao usuário para a análise. Além disso, também são apresentados os detalhes para a instalação e o uso das ferramentas. A principal contribuição deste relatório é o tutorial de instalação e uso, pois esse processo muitas vezes é custoso uma vez que a maioria das ferramentas possuem muitas dependências e configurações.

2 Ferramentas de monitoramento de desempenho

Nesta seção é apresentado o estudo realizado com algumas ferramentas de monitoramento de desempenho. Para cada uma delas é apresentado a técnica utilizada para a coleta de dados, a estrutura para o seu funcionamento e como os dados são dispostos ao usuário para a análise. Além disso, será abordado como essas ferramentas podem ser instaladas e utilizadas.

Ainda, vale ressaltar que embora a maioria das ferramentas que serão abordadas nesta seção possuam suporte para diferentes sistemas operacionais, todas as dicas de instalação e uso se referem ao sistema operacional Linux.

[1] Petaflópicos - Sistemas que podem executar 10^{15} operações de ponto flutuante por segundo.
[2] ExaFlop - 10^{18} operações de ponto flutuante por segundo.

2.1 Paraver

O Paraver (Pillet et al. 1995) é uma ferramenta de análise e visualização de dados que utiliza uma abordagem *offline*. Sua principal característica é a flexibilidade para representar arquivos de *trace* coletados por diferentes ferramentas, tais como Extrae (VI-HPS 2017a) e DIMENAS (VI-HPS 2017b).

A estrutura do Paraver (Figura 1) é divida em três módulos (BSC 2017a): filtro, semântico e representação. O filtro é o primeiro módulo que trabalha no arquivo de *trace*, registrando uma visão parcial dos dados. No módulo semântico, valores são computados baseados nos registros retornados pelo módulo de filtro. Como os arquivos de *trace* são obtidos através da técnica de rastreamento, eles possuem muitas informações, pois todos os eventos ocorridos durante a execução da aplicação são registrados. Logo, o módulo semântico pode ser compreendido como o mais importante, pois ele extrai e organiza os valores do arquivo de *trace* e os transfere para o módulo de representação. A representação das informações podem ser exibidas de três maneiras distintas (visualização, textual ou análise), as quais precisam ser interpretadas e relacionadas pelo usuário para a obtenção mais clara sobre o desempenho da aplicação.

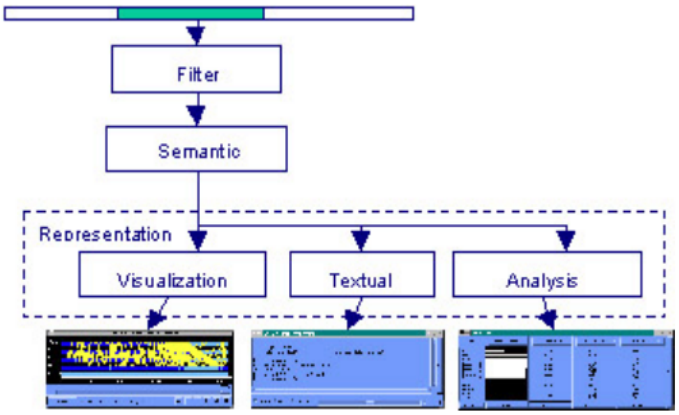

Figura 1: Arquitetura da ferramenta Paraver.
Fonte: (BSC 2017b)

Instalação

O Paraver possui suporte para diferentes sistemas operacionais, como Windows, Linux e MacOS X. Sua instalação é simples e é descrita abaixo.

Download da ferramenta:

`https://tools.bsc.es/downloads`

Descompactar o arquivo:

```
$ tar -jxvf nomedoarquivo.tar.bz2
```

Entre no diretório que foi criado e na pasta bin (`$ cd wxparaver/bin`) execute o comando:

```
$./wxparaver
```

Abrirá a interface de visualização do Paraver (Figura 2).

Para utilizar a ferramenta basta dispor de um arquivo de trace gerado pelas ferramentas de coleta Extrae ou Dimenas, carregar o arquivo no Paraver e gerar o resultado para análise através de um botão.

A Figura 3 representa o módulo de visualização, onde são apresentados os eventos ocorridos

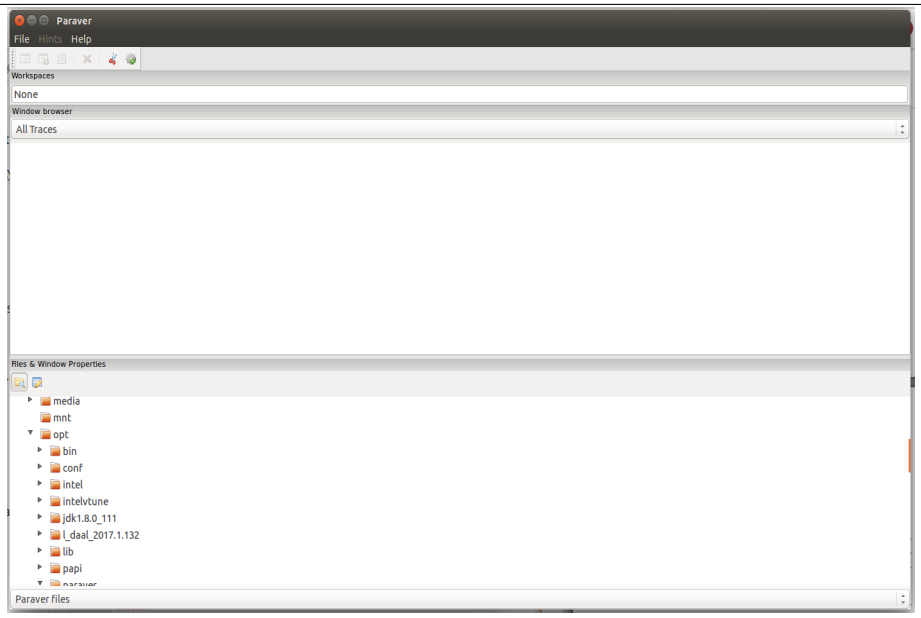

Figura 2: Interface da ferramenta Paraver.

em cada CPU envolvida na execução da aplicação bem como a comunicação entre eles. Embora a instalação e o uso do Paraver sejam simples, a interpretação de seus resultados não é trivial. O tipo de análise oferecido pela ferramenta é voltado para a fase de otimização dos códigos, uma vez que o Paraver auxilia na análise dos rastros da aplicação, avaliando como as funções estão sendo executadas e paralelizadas.

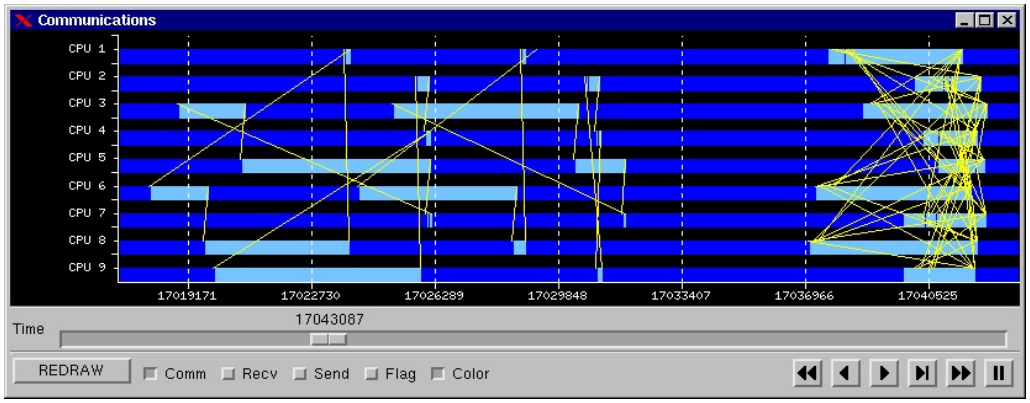

Figura 3: Módulo de visualização do Paraver.

2.2 VTune

O VTune (Intel 2015) é uma ferramenta de monitoramento desenvolvida pela Intel. A abordagem de monitoramento utilizada é a *online* e a coleta de dados é feita por amostragem e/ou rastreamento. A ferramenta dispõe de um conjunto de módulos (tais como *Basic Hotspots*, *Memory Access*, *CPU/GPU Concurrency*) que possibilitam diferentes tipos de análises de acordo com o parâmetro que se deseja monitorar. Por exemplo, com o módulo *Memory Access* é possível obter informações de como a aplicação está consumindo recursos relacionados à memória, tais como as atividades de carregamento (*load*) e armazenamento (*store*), a latência média de leituras por ciclo e a frequência em que a CPU fica aguardando a memória principal.

Instalação

A ferramenta VTune possui suporte para os sistemas operacionais Windows, Linux e MacOS. Suporta códigos em diferentes linguagens de programação (tais como C, C++, C, Fortran, Java, Python, Assembly, dentre outros) e paralelização (OpenMP, OpenCL e MPI). Para instalar a ferramenta é necessário realizar um cadastro no site da Intel (`https://registrationcenter.intel.com/en/forms/?productid=2993`), o qual irá requisitar o registro de um email. O link para download e a chave de ativação da ferramenta serão enviados para este endereço. Após o download da ferramenta, faz-se necessário alguns passos para a instalação, os quais são descritos abaixo.

Descompactar o arquivo:

```
$ tar -vzxf nomedoarquivo.tar.gz
```

Após descompactar o arquivo, será criado um novo diretório em `/opt`. Neste diretório execute o comando de instalação. Lembre-se de executá-lo como superusuário.

```
$ cd /opt/intel/vtune_amplifier_2018.2.0.551022
$ ./install.sh
```

Durante o processo de instalação a ferramenta dispõe de algumas configurações. É recomendável colocar as configurações default.

Para utilizar a ferramenta é necessário setar a variável de ambiente (amplxe-vars.sh) e executar o comando para abrir a interface do VTune:

```
$ source /opt/intel/vtune_amplifier_2018.2.0.551022/amplxe-vars.sh
$ amplxe-gui
```

Na Figura 4 é apresentada a interface, na qual o usuário escolhe o tipo de análise de acordo com os módulos disponíveis. Após terminar a execução da aplicação a interface apresenta algumas abas com o resultado da coleta, tais como *Summary* (Figura 5) e *Bottom up* (Figura 6). Nelas são apresentados, respectivamente, valores dos parâmetros coletados e o gráfico do desempenho de cada função que compõe o código da aplicação ao longo do tempo de execução. Outras abas também podem ser apresentadas, de acordo com o módulo escolhido pelo usuário ao executar a aplicação.

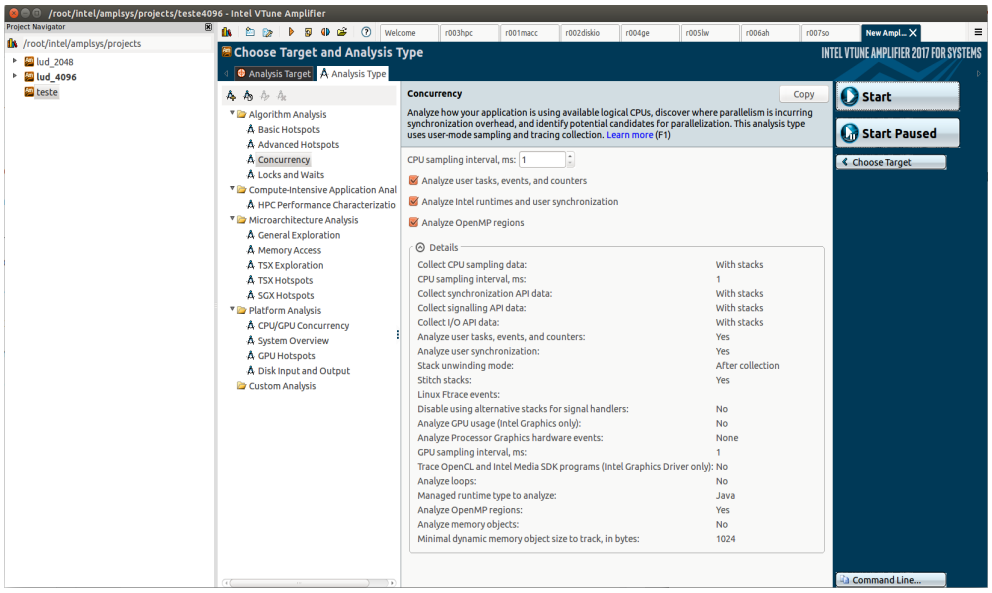

Figura 4: Interface Intel VTune - Tela Principal.

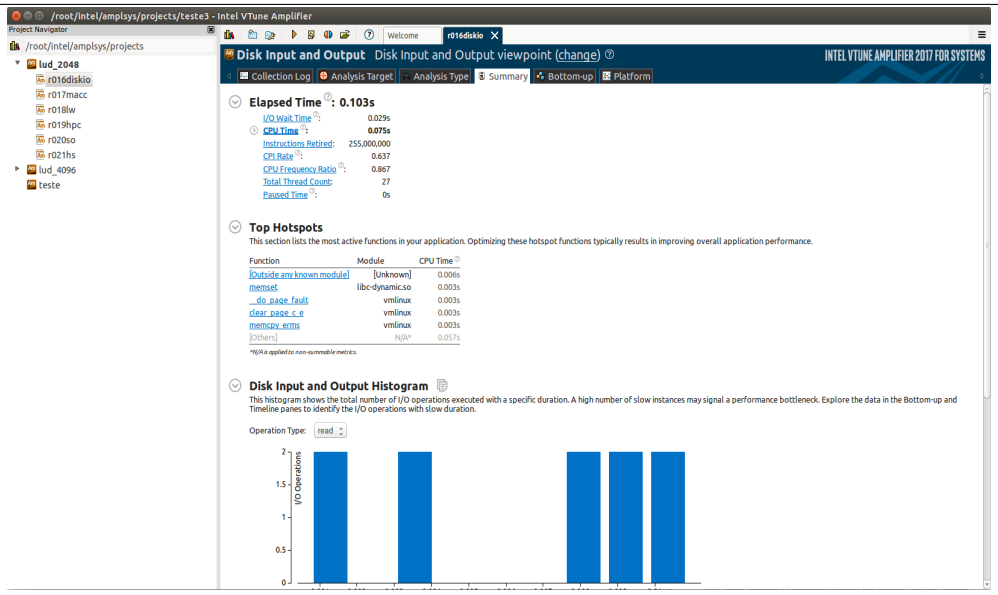

Figura 5: Interface Intel VTune - Summary.

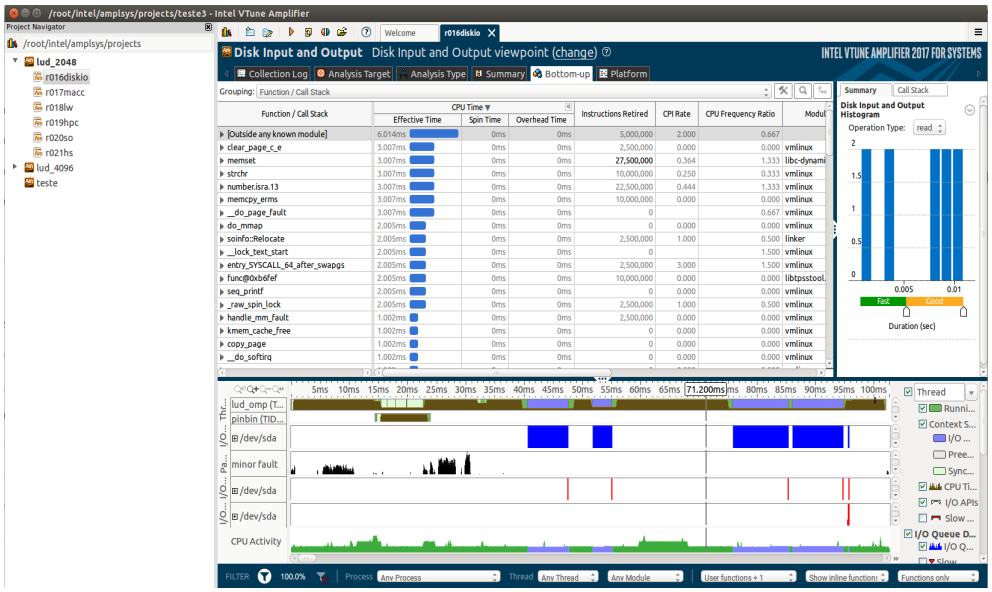

Figura 6: Interface Intel VTune - Bottom up.

Assim como a ferramenta Paraver (Seção 2.1), o tipo de análise oferecido pelo VTune é mais voltado para a otimização de códigos, pois a ferramenta oferece informações sobre cada função que compõe a aplicação, como apresentado na Figura 6. Além disso, muitos parâmetros de desempenho oferecidos pelo VTune possuem definições próprias, o que dificulta muito a interpretação dos resultados.

2.3 Pablo

O Pablo é um conjunto de ferramentas utilizado para instrumentar aplicações, coletar traços de execução e visualizar resultados (Reed et al. 1993). A ferramenta utiliza uma abordagem *offline*, na qual os dados são analisados após a execução da aplicação. O Pablo é composto por dois módulos principais, software de instrumentação e analisador de desempenho. O

software de instrumentação é formado por três peças chaves, apresentadas na Figura 7. São elas (Reed et al. 1992): *i)* interface gráfica, na qual o usuário pode interagir especificando os pontos de instrumentação do código, *ii)* modificadores (C ou Fortran), os quais geram o código fonte instrumentado, e *iii)* biblioteca, que captura os dados e gera os arquivos de *trace*. O analisador de desempenho é formado por um conjunto de módulos de transformação de dados, oferecendo ao usuário análise gráfica dos dados coletados (Reed et al. 1993).

A ferramenta Pablo possui um formato padronizado, *Self-Defining Data Format* (SDDF), no qual os dados são representados. Os componentes do módulo de instrumentação produzem arquivos de *traces* em SDDF e os componentes do módulo de análise utilizam esses arquivos como entrada e os representa graficamente. Ainda que a ferramenta possua um formato padronizado, seu analisador de desempenho pode ser utilizado por arquivos de *traces* gerados por outras bibliotecas. Pablo oferece conversores, os quais transformam outros formatos para o padrão SDDF.

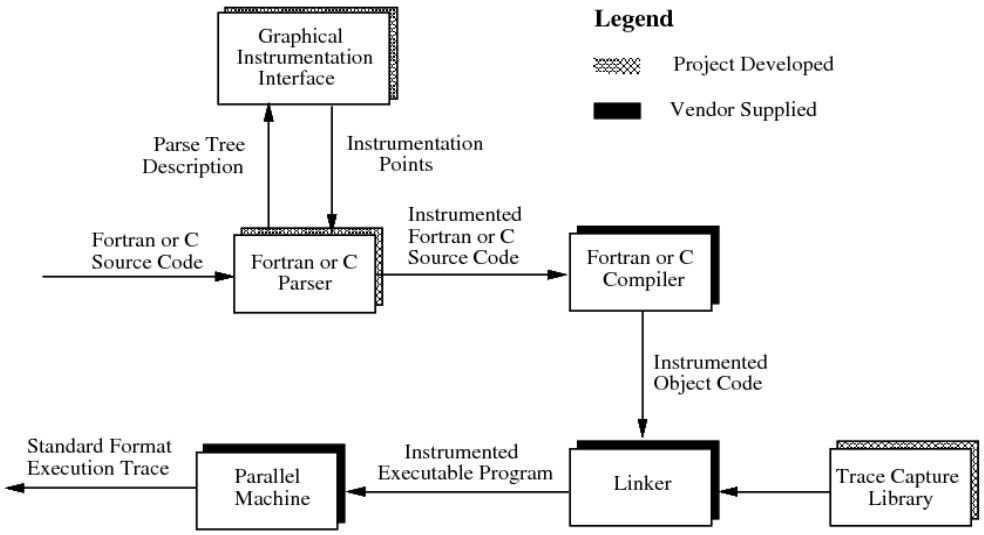

Figura 7: Componentes do software de instrumentação do Pablo.

Instalação

A ferramenta está disponível para download em `http://wotug.org/parallel/performance/tools/pablo/`

Descompactar o pacote:

```
tar -xzvf PabloSrc.tar.gz
```

O próximo passo é editar o arquivo "Makefile.defines"que está contido no diretório criado após a descompactação. Siga os passos descritos nos comentários desse arquivo para definir as configurações de acordo com a sua necessidade. Quando terminar as alterações necessárias, crie os executáveis. Esse processo é realizado executando o comando:

```
$ gnumake all
```

Lembrando que o Pablo utiliza um formato padronizado, SDDF, no qual os dados são representados. Os conversores de outros formatos para o SDDF são executáveis que estão localizados no diretório criado após descompactar a ferramenta. Para instalar os executáveis e as bibliotecas necessárias digite o comando:

```
$ gnumake install
```

Após a conclusão da compilação, vá para o diretório `.../Visual/Src/System/Build` e exe-

cute o comando para abrir a interface da ferramenta Pablo (Figura 8):

```
$ ./runPablo
```

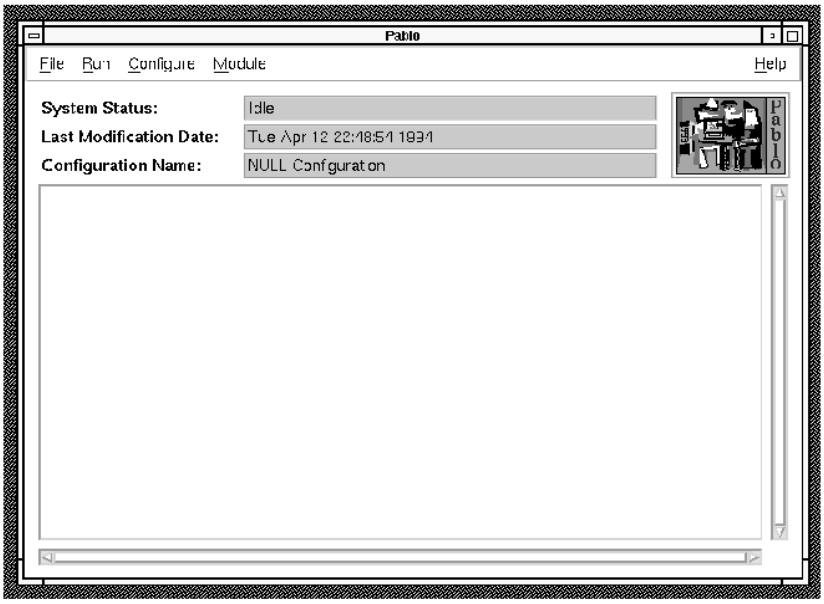

Figura 8: Interface inicial da ferramenta Pablo.

Para a visualização dos dados de desempenho, o usuário informa qual arquivo SDDF será analisado, quais módulos de transformação de dados serão utilizados e quais métricas deseja visualizar graficamente. Esses módulos interagem entre si e calculam a métrica desejada a partir da extração de dados do arquivo de *traces*. A Figura 9 mostra o gráfico de análise com a utilização do módulo de perfil. Esse módulo calcula a distribuição de ocorrências de eventos e os intervalos para cada evento, e exibe esses dados em forma tabular e gráfica (Reed 1994).

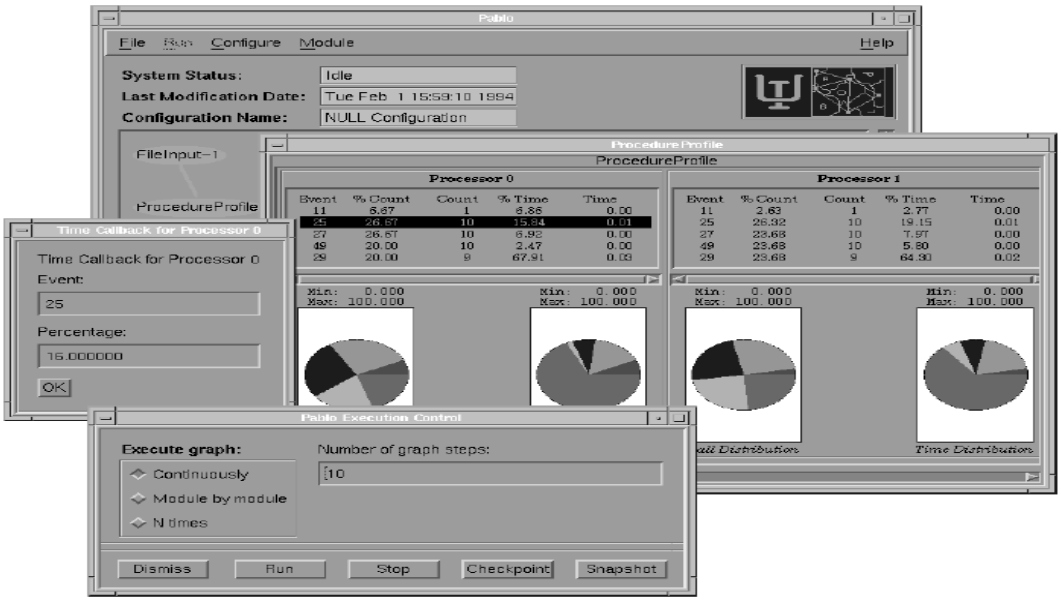

Figura 9: Interface de análise dos resultados da ferramenta Pablo.

Embora seja possível obter diferentes métricas com o Pablo, tais como o tempo de chamada por processador e o percentual de duração de cada evento ocorrido, a ferramenta possui

muitas configurações de usuário para a geração de gráficos, o que torna o processo de análise demorado para experimentos que muitas vezes deveriam ser rápidos e simples de serem realizados.

2.4 TAU

O TAU (Shende and Malony 2006) é um conjunto de ferramentas que utiliza as técnicas de amostragem e rastreamento para monitorar a execução de aplicações implementadas em Fortran, C, C++, Java e Python. A arquitetura da ferramenta é apresentada na Figura 10 e possui três estágios: instrumentação da aplicação, medição (ou coleta) de dados e visualização de resultados.

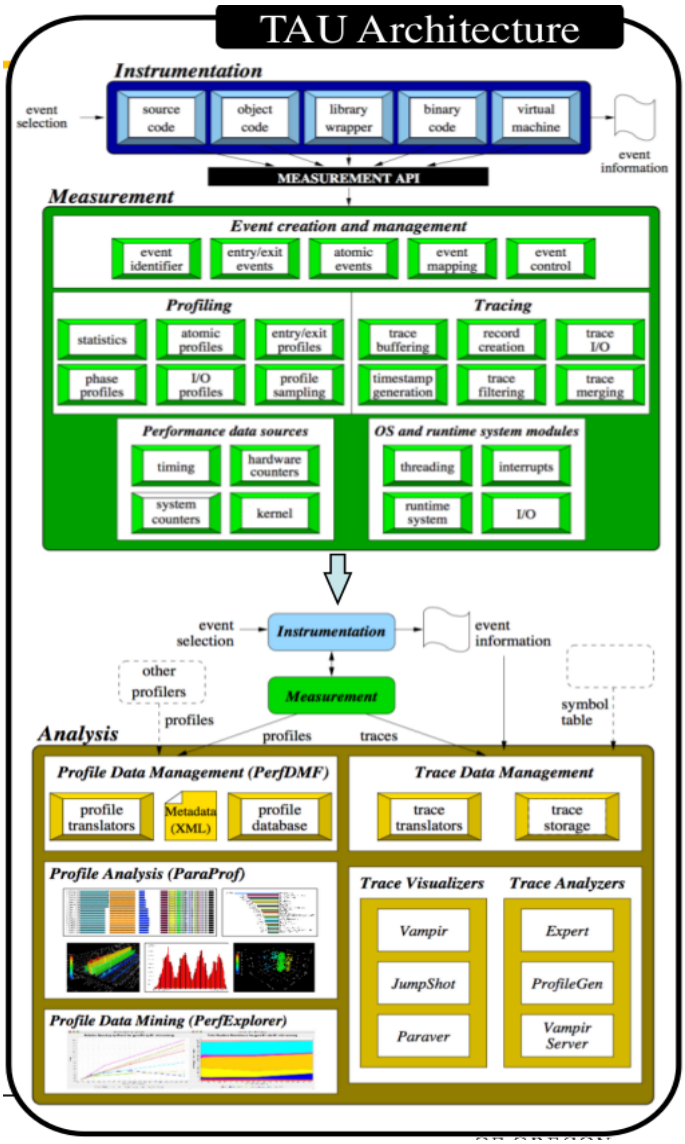

Figura 10: Arquitetura da ferramenta TAU.

A instrumentação do código consiste em inserir chamadas para a *Application Programming Interface* (API) de medição da ferramenta, podendo ser realizada utilizando a biblioteca *tau_exec* ou através de diretivas de compilação (Shende and Malony 2006). Cada método de instrumentação possui diferentes requisitos. Por exemplo, com a utilização da *tau_exec* há necessidade de recompilar o código da aplicação após inserir as diretivas de instrumen-

tação, enquanto que utilizando as diretivas do compilador não há necessidade de realizar este procedimento. Ainda, de acordo com a instrumentação realizada, diferentes parâmetros para análise são coletados. Com isso, dependendo dos dados que se deseja coletar em um experimento, pode ser necessário a utilização de mais de um método de instrumentação.

Os dados de rastreamento e de amostragem são coletados em arquivos separados. Assim, TAU oferece um conjunto de ferramentas para os dados obtidos por meio da amostragem, tais como *Paraprof* (Bell et al. 2003) e *PerfExplorer* (Huck and Malony 2005). A ferramenta *pprof*, na qual os dados são apresentados em forma textual, também está disponível para analisar dados obtidos por amostragem. Entretanto, ainda não há uma ferramenta para visualização dos dados coletados por rastreamento. TAU apenas exporta os parâmetros coletados para formatos que são compatíveis com outras ferramentas, tais como JumpShot (Zaki et al. 1999), Vampir (Nagel et al. 1996) e Paraver (Seção 2.1). Na Figura 11 é apresentada a interface da ferramenta Paraprof, na qual as funções são representadas por histogramas estatísticos, os quais são codificados por cores, indicando os eventos ocorridos.

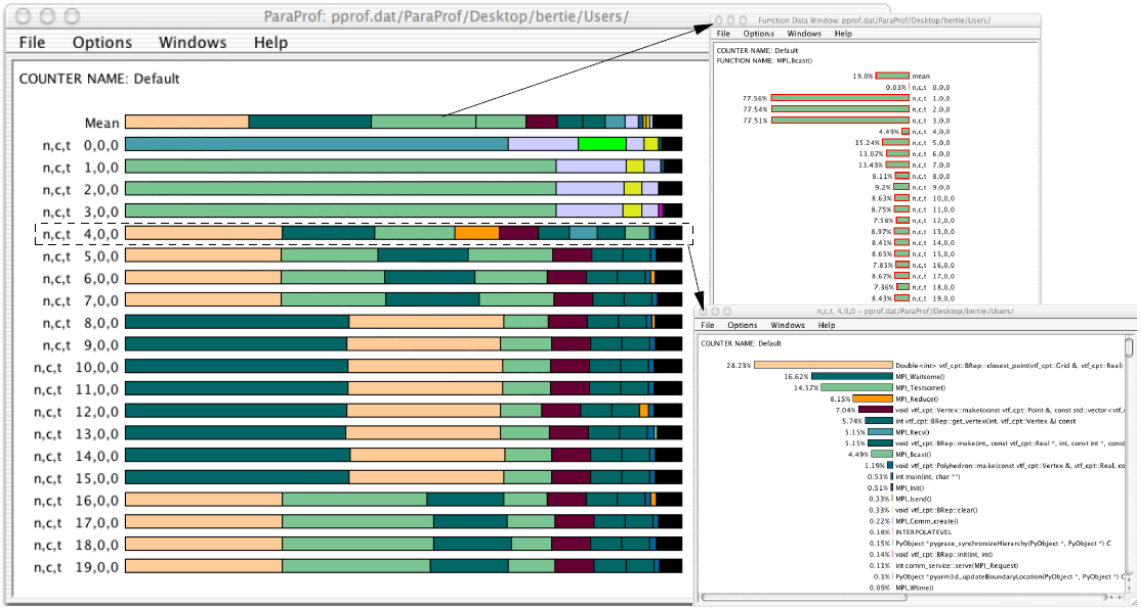

Figura 11: Paraprof - Interface de visualização de dados da ferramenta TAU.
Fonte: (Bell et al. 2003)

Instalação

O download da ferramenta está disponível em `https://www.cs.uoregon.edu/research/tau/downloads.php`. Após o download, descompacte o arquivo, entre no diretório criado e execute o comando de instalação.

```
$ tar -xvzf nomedoarquivo.tgz
$ cd tau-2.27
$ ./installtau
```

O TAU oferece uma interface gráfica de configuração. Essas configurações incluem tipo de compilador, biblioteca de MPI, entre outras. Para abrir a interface e realizar a configuração de acordo com sua necessidade basta digitar o comando abaixo:

```
$ ./tau_setup
```

Após as configurações instale as bibliotecas:

```
$ make clean install
```

Para informações de como utilizar os scripts de instrumentação fornecidos pela ferramenta TAU acesse o conteúdo disponível em `/tau-2.27/INSTALL`.

Como apresentado na Figura 11, o TAU oferece um conjunto de estatísticas sobre o desempenho da aplicação. Porém, se a instrumentação for realizada através do compilador, o usuário necessita de bons conhecimentos de compilação do *kernel* do Linux (Fernandes 2003). Este sistema operacional é amplamente utilizado nos ambientes HPC.

2.5 PAPI

O PAPI (Browne et al. 2000) é uma ferramenta desenvolvida com o objetivo de acessar os contadores de desempenho de hardware encontrados na maioria dos processadores modernos (Dongarra et al. 2001). Os contadores de hardware são capazes de medir diferentes aspectos do funcionamento de um processador, tais como falhas de acesso a memória cache (*cache miss*), quantidade de instruções executadas e número de execuções de uma instrução em particular. Realizando essas medições enquanto a aplicação está sendo executada, o usuário obtém a relação entre os eventos de hardware e o desempenho da aplicação.

O PAPI fornece duas interfaces para acessar os contadores de hardware (Dongarra et al. 2001): i) simples e de alto nível, para a aquisição de medidas simples; e ii) totalmente programável e de baixo nível, voltada para usuários com necessidades mais sofisticadas. A interface de baixo nível gerencia eventos de hardware definidos pelo usuário, denominados *EventSets* (Papi 2016). A interface de alto nível fornece a capacidade de iniciar, parar e ler os contadores, para uma lista predefinida de eventos. Essa lista está disponível no arquivo *papiStdEventDefs.h* na distribuição PAPI.

Instalação

Para utilizar a ferramenta é necessário instalar seus componentes e suas bibliotecas. A distribuição completa do PAPI está disponível no github:

```
$ git clone https://bitbucket.org/icl/papi.git
```

Para que a versão do PAPI sempre esteja atualizada com o repositório github, execute os comandos abaixo:

```
$ cd papi
$ git pull https://bitbucket.org/icl/papi.git
```

Entre no diretório `$ papi/src` para dar início ao processo de configuração.

```
$ cd papi/src
$ ./configure
$ make
```

Existem muitas opções que podem ser configuradas via linha de comando, as quais podem variar dependendo da versão do PAPI. Para detalhes completos sobre essas opções use:

```
$ ./configure -help
```

Entre em `ctests/zero` e execute um conjunto de testes para saber quais as funcionalidades do PAPI que são compatíveis com a arquitetura do hardware. A medida que cada teste é realizado, a ferramenta retornará um status (*PASSED, FAILED, ou SKIPPED*). Vale lembrar que os testes são *SKIPPED* se a funcionalidade que está sendo testada não é suportada por essa plataforma.

```
$ cd ctests/zero
```

```
$ make test
$ make fulltest
$ ./run_tests.sh -v
```

Ainda no diretório `papi/src` instale as bibliotecas do PAPI, os manuais e os programas de teste de compilação.

```
$ make install
$ make install-man
$ make install-tests
```

Após criar as bibliotecas, instale os componentes necessários para o funcionamento da ferramenta. Os componentes estão localizados em `src/components/` e a instalação de cada um deles está em seu arquivo README. Antes de executar qualquer componente que exija configuração, o script de configuração para esse componente deve ser executado para gerar o Makefile que contém as definições adequadas. Normalmente, o script só precisará ser executado uma vez. Ainda, caso o programador deseje adicionar novas funcionalidades ao PAPI será necessário criar um novo componente para a ferramenta. Instruções básicas sobre como criar um novo componente podem ser encontradas em `src/components/README`.

Eventos de hardware podem não ser os mesmos em diferentes arquiteturas, e nem todos os eventos são possíveis de serem coletados em todas as arquiteturas. No entanto, PAPI mapeia cada um dos eventos das diferentes arquiteturas para eventos nativos, facilitando a interpretação do usuário (Browne et al. 2000). É possível monitorar um conjunto de eventos sobre a execução da aplicação em diferentes arquiteturas e o tipo de análise realizada pela ferramenta é mais voltada para quando o foco é melhorar o código da aplicação. Ou ainda, quando o objetivo é analisar a própria arquitetura do processador para a fabricação de gerações de processadores melhores.

2.6 HPCToolkit

O HPCToolkit (HPCToolkit 2016) é um conjunto de ferramentas para medição, monitoramento e análise de desempenho de aplicações. Para a coleta de dados, a ferramenta faz uso de duas diferentes técnicas de amostragem (guiada pelo tempo e guiada por eventos). A amostragem guiada pelo tempo é aquela utilizada pela maioria das ferramentas de monitoramento, a qual consiste em coletar, em intervalos fixos, o estado quantitativo do consumo de diferentes recursos computacionais pela aplicação. A amostragem guiada por eventos é um tipo especial de coleta e se refere ao período de amostragem expresso pelo número de ocorrência de eventos (Mellor-Crummey et al. 2012). Este tipo de amostragem requer instrumentação para acessar os contadores de eventos. O HPCToolkit instrumenta o código binário da aplicação e os contadores são acessados através da ferramenta PAPI (Seção 2.5).

A Figura 12 (Adhianto et al. 2010) ilustra os principais componentes da ferramenta e como eles estão relacionados. No primeiro estágio, para a geração do código binário otimizado, é necessário compilar e fazer o link da aplicação com o HPCToolkit. A partir da amostragem guiada pelo tempo, dados de desempenho são gerados pelo *hpcrun*, o qual constrói o perfil da aplicação. Paralelo a esse estágio, através da amostragem guiada por eventos, o componente *hpcstruct* analisa o binário da aplicação para obter mais informações sobre a execução. A partir da combinação e interpretação das informações produzidas por *hpcrun* e *hpcstruct*, o componente *hpcprof* constrói um banco de dados de desempenho correlacionando a estrutura do código fonte com as medições. Por fim, com base no banco de dados de desempenho, as ferramentas gráficas *hpcviewer* e *hpctraceviewer* apresentam os dados para análise.

Instalação

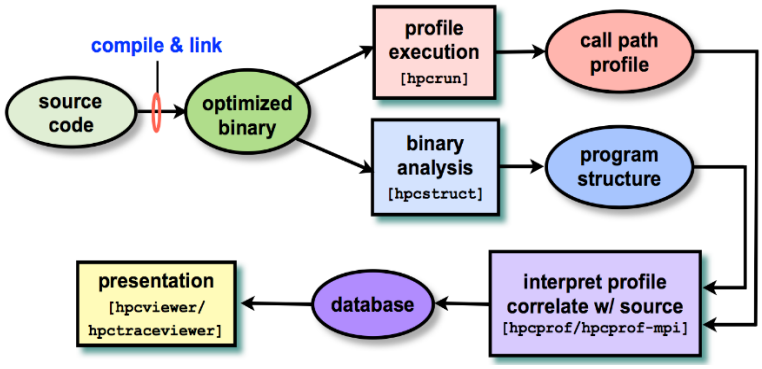

Figura 12: Arquitetura da ferramenta HPCToolkit.

Download da ferramenta:

```
$ git clone https://github.com/HPCToolkit/hpctoolkit.git
```

Faça download também do hpctoolkit-externals. Pense no hpctoolkit-externals como um gerenciador de pacotes que automaticamente configura e constrói bibliotecas de suporte que o HPCToolkit precisa.

```
$ git clone https://github.com/HPCToolkit/hpctoolkit-externals.git
```

Faça o download das versões binárias do hpcviewer e do hpctraceviewer para a plataforma na qual você está construindo o HPCToolkit. O download está disponível em `http://hpctoolkit.org/download/hpcviewer/`.

É necessário utilizar o comando de configuração do HPCToolkit para verificar automaticamente a disponibilidade das dependências no sistema Linux utilizado.

```
$ ./configure && make && make install
```

Instalar e configurar o hpctoolkit-external:

```
$ cd hpctoolkit-externals
$ mkdir BUILD && cd BUILD
$ ../configure [CC=<c-compiler>] [CXX=<c++-compiler>]  [-prefix=<hpctoolkit-externals-
install>]
$ make install
$ make clean
```

Para especificar um compilador C e C++ diferente de gcc e g++, use as variáveis CC e CXX para inserir o nome do compilador.

Para instalar e configurar o hpctoolkit-external execute os comandos abaixo. Novamente, para especificar um compilador C e C++ diferente do gcc e g++, use as variáveis CC e CXX para inserir o nome do compilador. Além disso, para ativar o suporte ao MPI garanta que um mpicxx ou mpiCC esteja no caminho do seu shell ou especifique o caminho na variável MPICXX. Para ativar medições baseadas em PAPI, use a opção `-with-papi` e especifique o caminho da sua instalação PAPI.

```
$ cd hpctoolkit
$ mkdir BUILD && cd BUILD
$ ../configure [CC=<c-compiler>] [CXX=<c++-compiler>] [MPICXX=<mpi-c++-compiler>]
-prefix=<hpctoolkit-install>  -with-externals=<hpctoolkit-externals-install>  [-
with-papi=<papi-install>]
```

```
$ make install
```

Para instalar hpcviewer e hpctraceviewer, descompacte os arquivos .tgz baixados e execute os comandos abaixo. Em `<hpcviewer>` e `<hpctraceviewer>` coloque os diretórios criados pela descompactação dos respectivos pacotes. Além disso, no lugar de `<hpctoolkit-install>` coloque o diretório de instalação do HPCToolkit. Realizadas todas essas instalações e configurações, a ferramenta HPCToolkit está pronta para o uso.

```
$ tar xvzf hpcviewer-linux.gtk.x86_64.tgz
$ tar xvzf hpctraceviewer-linux.gtk.x86_64.tgz
$ cd <hpcviewer> && ./install [ -j <path-to-java> ] <hpctoolkit-install>
$ cd <hpctraceviewer> && ./install [ -j <path-to-java> ] <hpctoolkit-install>
```

Para utilizar HPCToolkit é necessário inserir suas diretivas de utilização ao executar a aplicação, como no exemplo da Figura 13.

Listing 1: Creating the code structure

```
hpcstruct ./cg.C.x
```

Listing 2: Running the application, event-based sampling

```
OMP_NUM_THREADS=4 hpcrun -e PAPI_TOT_CYC@5000000 \
-e PAPI_L1_DCM@40000 ./cg.C.x
```

Listing 3: Running the application, tracing

```
OMP_NUM_THREADS=4 hpcrun -t -e WALLCLOCK@5000 ./cg.C.x
```

Listing 4: Correlating with source code

```
hpcprof -S cg.C.x.hpcstruct -I ../'*' hpctoolkit-cg.C.x-measurements
```

Listing 5: View results

```
hpcviewer hpctoolkit-cg.C.x-database
```

Figura 13: Compilando a aplicação com diretivas da ferramenta HPCToolkit.

Os parâmetros de desempenho possíveis de serem analisados com HPCToolkit dependem das diferentes interfaces de visualização. A interface *hpcviewer* (Figura 14), composta por dois painéis principais, exibe o código fonte da aplicação no painel superior e associa esse código à uma tabela de métricas de desempenho no painel inferior. Enquanto que a interface *hpctraceviewer* (Figura 15), fornece análise gráfica da ocorrência de eventos em cada *thread* da aplicação. Cada cor distinta nas linhas de tempo representa diferentes eventos.

Na maioria das ferramentas que oferecem dados de desempenho em mais baixo nível (granularidade fina), há necessidade de instrumentar o código fonte da aplicação, o que nem sempre é uma tarefa de baixa complexidade. Além disso, normalmente essas ferramentas, tais como o Paraver (Seção 2.1), o TAU (Seção2.4) e o Intel VTune (Seção 2.2), fazem uso da técnica de rastreamento para coletar os dados, considerada a abordagem que gera o maior *overhead* na análise. Com HPCToolkit, o usuário obtém informações semelhantes às obtidas por essas diferentes ferramentas. Porém, o diferencial é que não há necessidade de instrumentar o código fonte da aplicação. Através da amostragem guiada por eventos, a ferramenta instrumenta apenas o código binário da aplicação. Além disso, esse tipo de amostragem, possui baixo *overhead* quando comparada com o rastreamento. No entanto, o tipo de coleta realizada pela ferramenta é mais voltada para quando o objetivo é otimizar as aplicações, por envolver a interação entre chamadas e funções do código.

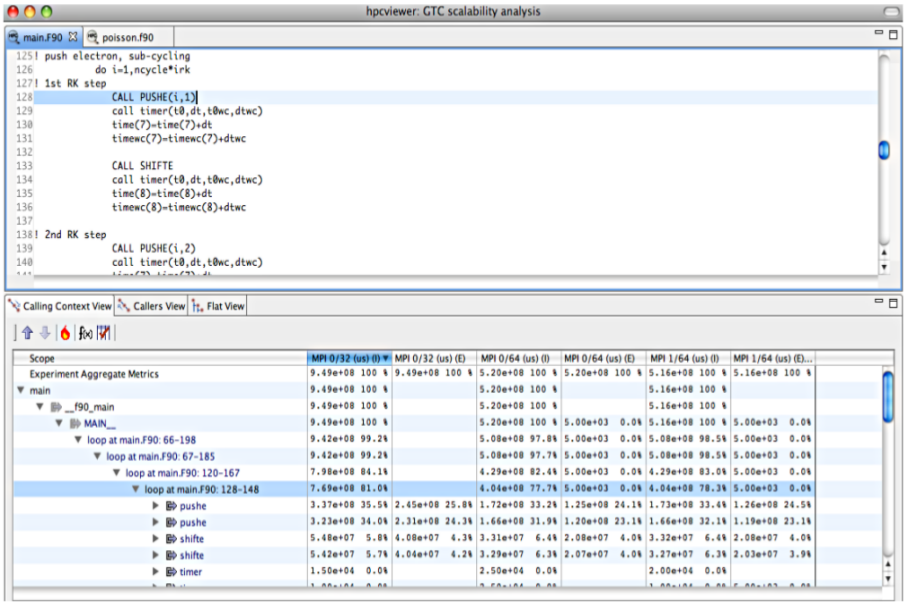

Figura 14: HPCToolkit - Interface *hpcviewer*.

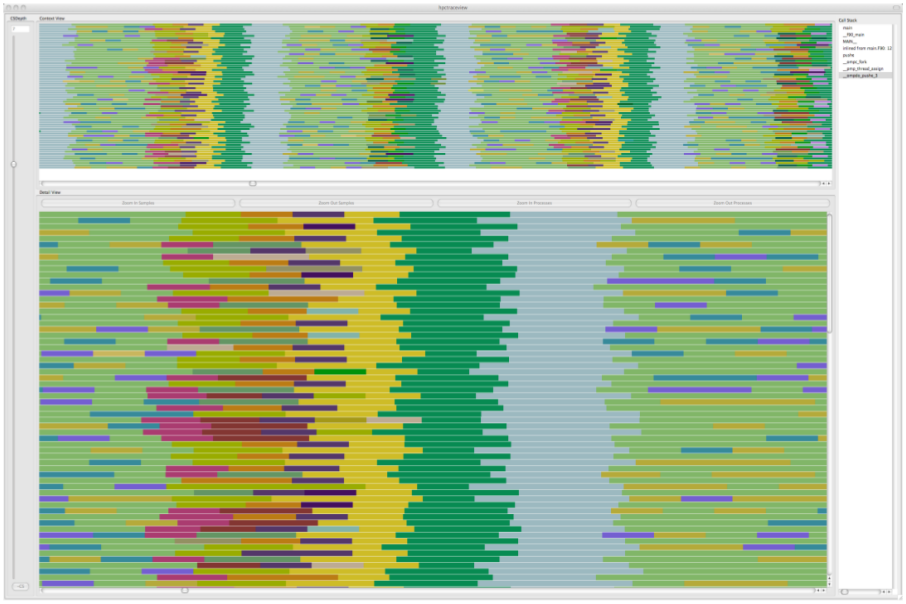

Figura 15: HPCToolkit - Interface *hpctraceviewer*.

3 Ferramentas de monitoramento de energia

Nesta seção é apresentado o estudo realizado com algumas ferramentas de monitoramento de energia. Para cada uma delas é apresentado a técnica utilizada para a coleta de dados, a estrutura para o seu funcionamento e como os dados são dispostos ao usuário para a análise. Além disso, será abordado como essas ferramentas podem ser instaladas e utilizadas.

Ainda, vale ressaltar que embora a maioria das ferramentas que serão abordadas nesta seção possuam suporte para diferentes sistemas operacionais, todas as dicas de instalação e uso se referem ao sistema operacional Linux.

3.1 PowerAPI

PowerAPI (PowerAPI 2016) foi desenvolvida com o objetivo de oferecer ao usuário uma maneira simples e eficiente de estimar o consumo de energia, sem a necessidade de componente externo para a medição. A partir do acesso aos sensores, com intervalos de amostras definidos pelo usuário e com o Identificador de Processo (PID) da aplicação, a ferramenta estima o consumo energético por componentes, tais como CPU, memória e disco (Bourdon et al. 2013). Há uma versão compilada da ferramenta para ser utilizada via linha de comando, e outra para ser utilizada como biblioteca por outras ferramentas.

A arquitetura do PowerAPI (Figura 16) é dividida em dois diferentes grupos de módulos: os que coletam informações dos sensores e os que calculam o consumo energético. Os módulos de coleta, identificados na Figura 16 como *Sensors*, podem ser iniciados ou interrompidos durante o tempo de execução da aplicação (Noureddine et al. 2012b). Enquanto que os módulos *Formulae* obtém as informações coletadas e estima, com o auxilio de fórmulas matemáticas, o consumo energético da aplicação. As fórmulas utilizadas para o cálculo do consumo energético são descritas em (Noureddine et al. 2012a). Ainda, um banco de dados é utilizado para armazenar informações estatísticas e para calibrar a ferramenta. Todos os componentes da PowerAPI são gerenciados através de um módulo especial (*Lice cycle manager*), o qual permite o usuário iniciar, parar, adicionar, remover ou até mesmo modificar os outros módulos, dependendo de sua necessidade no monitoramento.

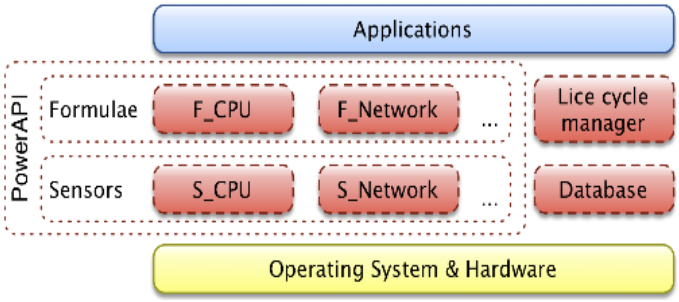

Figura 16: Arquitetura da PowerAPI.

Instalação

O download da ferramenta está disponível em `https://github.com/Spirals-Team/powerapi/wiki/Getting-started-CLI`

Descompacte o arquivo:

```
$ tar -xzvf nomedoarquivo.tar.gz
```

Atualmente, as dependências da ferramenta são: 1) java e 2) *Thermal Design Power (TDP)* do processador.

Instalação do java:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

O TDP do processador deve ser inserido no arquivo de configuração `powerapi.conf`.

```
$ cd PowerAPI/conf
$ vi powerapi.conf
```

Para que PowerAPI inicie o monitoramento é necessário informar o PID do processo/aplicação

que será monitorado. Como mencionado, com o PowerAPI é possível coletar a energia consumida por componentes. Um exemplo de como utilizar a ferramenta para coletar o consumo energético da CPU pode ser visualizado abaixo.

```
$ ./bin/powerapi modules procfs-cpu-simple monitor -frequency 500 -pids 2356 -
console
```

Para informações de como utilizar os outros módulos disponíveis pela ferramenta acesse `https://github.com/Spirals-Team/powerapi/wiki/Modules`.

A visualização dos dados coletados por PowerAPI é ilustrada na Figura 17. Neste experimento, a ferramenta foi utilizada para monitorar o consumo de energia da CPU para a execução da aplicação Decomposição LU (matriz de 4096x4096) da suíte Rodinia (Rodinia 2015). As amostras do consumo de energia são expostas em tempo de execução linha após linha.

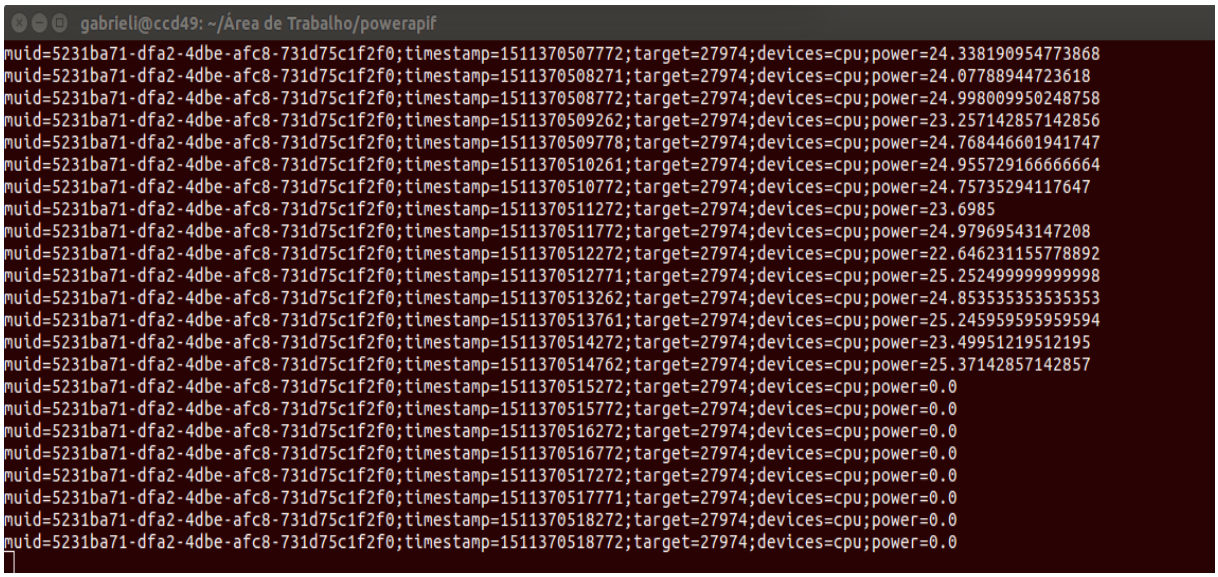

Figura 17: Visualização da coleta com PowerAPI.

Como mencionado, para monitorar um processo específico, a ferramenta necessita do PID do processo. No entanto, como o PID é criado somente ao executar a aplicação, não há possibilidade de obter o consumo de potência no instante inicial ou alguns instantes antes da execução da aplicação, o que é importante na análise para estabelecer qual a energia consumida, individualmente, pelo sistema e pela aplicação (Silva et al. 2017). Além disso, para melhor visualização dos resultados, o ideal seria a ferramenta oferecer ao usuário a análise gráfica dos dados, ao longo do tempo de execução da aplicação. Visualizar os dados graficamente facilita a avaliação e a interpretação dos experimentos.

3.2 RAPL

Os recentes chips da *Intel Sandy Bridge* incluem a interface RAPL (RAPL 2016), a qual consiste de um software (ou sensor) com a capacidade de monitorar, controlar e receber notificações sobre o consumo energético (McCraw et al. 2014). A estimativa do consumo de energia realizada pelo RAPL é calculada a partir de modelos.

O RAPL facilita a coleta sobre o consumo energético no início da execução de uma aplicação, pois a taxa de atualização das leituras realizadas pela ferramenta ocorrem com uma frequência de atualização de 1 milissegundo (Weaver et al. 2013). Além disso, outras ferramentas utilizam o RAPL para coletar dados de energia, como por exemplo a ferramenta

PAPI (Seção 2.5) a partir da versão 5.0.0.

Instalação

Existem três maneiras distintas para acessar os resultados obtidos pelo RAPL: i) lendo os arquivos em `/sys/class/powercap/intel-rapl/intel-rapl:0` utilizando a ferramenta PowerCap; ii) utilizando a interface `perf_event`; ou iii) Através do acesso aos registradores MSRs em `/dev/msr`.

Como mencionado, os sensores RAPL estão presentes na arquitetura Intel. Porém, nessa arquitetura as coletas de medidas sobre o consumo de energia só são possíveis no modo *kernel* privilegiado (Hähnel et al. 2012). O fator limitante para a utilização do RAPL é justamente este, pois é necessário permissão a nível privilegiado para realizar as leituras. Este tipo de permissão nem sempre é disponibilizado em ambientes de HPC controlados como o supercomputador Santos Dumont (SDumont) (SINAPAD 2014), o que dificultou obter maiores detalhes sobre como explorar o RAPL.

3.3 NVIDIA System Management Interface (NVIDIA-SMI)

NVML (NVIDIA 2012) é uma API baseada na linguagem C, desenvolvida para monitorar e gerenciar estados de dispositivos NVIDIA GPU, tais como, taxa de utilização de GPU, execução de processo, estado e desempenho de clock, temperatura e velocidade, consumo e gerenciamento de energia. O NVIDIA-SMI (NVIDIA 2017) é um programa de linha de comando que permite aos administradores do sistema consultar, com os privilégios apropriados, os estados dos dispositivos GPU. Suas funções são fornecidas pela biblioteca NVML. O NVIDIA-SMI é voltado para os produtos TeslaTM, GRIDTM, QuadroTM e Titan X, embora com suporte limitado também esteja disponível em outras GPUs NVIDIA.

Instalação

Antes de instalar o NVIDIA-SMI é necessário instalar o driver R390 mais recente da NVIDIA, o qual está disponível para download em `www.nvidia.com/drivers`.

O download da ferramenta está disponível em `https://developer.nvidia.com/cuda-downloads`. Sua instalação é simples e não requer muitas dependências, apenas o driver R390.

Ao acessar o link de download será solicitado que o usuário informe o sistema operacional no qual a ferramenta será instalada, além da distribuição, da versão e da arquitetura. O NVIDIA-SMI está disponível em diferentes tipos de instalação, tais como *deb(local), deb(network) e runline(local)*. Cada uma dessas opções tem o seu pacote para download e diferentes passos para a instalação. Por exemplo, ao escolher a opção *deb(local)*, é necessário fazer download do arquivo que aparecerá na tela e realizar o conjunto de comandos listados abaixo:

```
$ sudo dpkg -i cuda-repo-ubuntu1604-9-1-local_9.1.85-1_amd64.deb
$ sudo apt-key adiciona / var / cuda-repo- <versão> / 7fa2af80.pub
$ sudo apt-get update
$ sudo apt-get install cuda
```

As métricas coletadas pelo NVIDIA-SMI (Figura 18) podem ser visualizadas pelo usuário via linha de comando ou as saídas podem ser enviadas para arquivos em formato *Comma-separated values* (CSV) e *Extensible Markup Language* (XML).

```
==============NVSMI LOG==============

Timestamp                           : Fri Jun 22 11:32:26 2012

Driver Version                      : 295.45

Attached GPUs                       : 1

GPU 0000:01:00.0
    Product Name                    : Tesla C2070
    Display Mode                    : Enabled
    Persistence Mode                : Disabled
    Driver Model
        Current                     : N/A
        Pending                     : N/A
    Serial Number                   : 0322211066758
    GPU UUID                        : GPU-ed89e2d3-e674-1a30-8dd0-eaee6789ee19
    VBIOS Version                   : 70.00.44.00.02
    Inforom Version
        OEM Object                  : 1.0
        ECC Object                  : 1.0
        Power Management Object     : 1.0
    PCI
        Bus                         : 0x01
        Device                      : 0x00
        Domain                      : 0x0000
        Device Id                   : 0x06D110DE
        Bus Id                      : 0000:01:00.0
        Sub System Id               : 0x077210DE
        GPU Link Info
            PCIe Generation
                Max                 : 2
                Current             : 2
            Link Width
                Max                 : 16x
                Current             : 16x
    Fan Speed                       : 30 %
    Performance State               : P0
    Memory Usage
        Total                       : 5375 MB
        Used                        : 9 MB
        Free                        : 5365 MB
    Compute Mode                    : Default
    Utilization
        Gpu                         : 0 %
        Memory                      : 0 %
    Ecc Mode
        Current                     : Enabled
        Pending                     : Enabled
    ECC Errors
        Volatile
            Single Bit
                Device Memory       : 0
                Register File       : 0
                L1 Cache            : 0
                L2 Cache            : 0
                Total               : 0
            Double Bit
                Device Memory       : 0
                Register File       : 0
                L1 Cache            : 0
                L2 Cache            : 0
                Total               : 0
        Aggregate
            Single Bit
                Device Memory       : 0
                Register File       : 0
                L1 Cache            : 0
                L2 Cache            : 0
                Total               : 0
            Double Bit
                Device Memory       : 0
                Register File       : 0
                L1 Cache            : 0
                L2 Cache            : 0
                Total               : 0
    Temperature
        Gpu                         : 52 C
    Power Readings
        Power Management            : N/A
        Power Draw                  : N/A
        Power Limit                 : N/A
    Clocks
        Graphics                    : 573 MHz
        SM                          : 1147 MHz
        Memory                      : 1494 MHz
    Max Clocks
        Graphics                    : 573 MHz
        SM                          : 1147 MHz
        Memory                      : 1494 MHz
    Compute Processes               : None
```

Figura 18: Parâmetros coletados pela ferramenta NVIDIA-SMI.

3.4 VTune

A ferramenta Intel VTune, abordada na Seção 2.2, além de monitorar desempenho também coleta medidas sobre o consumo de energia. Para isso, a ferramenta faz uso de uma abordagem de medição direta utilizando sensores internos. No entanto, para monitorar energia, VTune necessita de um *driver* específico (*powerdk*). Foram feitas inúmeras tentativas de instalação deste *driver*, sem sucesso. Além disso, no ambiente SDumond ele também não está disponível, o que inviabilizou a sua instalação e o seu uso para apresentar maiores detalhes.

4 Conclusão

Como mencionado, por meio do monitoramento e da avaliação de desempenho das aplicações científicas é possível caracterizar, para os diferentes modelos e tamanhos de problema, quais seus principais requisitos computacionais, como por exemplo CPU, E/S e memória, e qual a relação desses requisitos com o consumo de energia. Porém, cada ferramenta coleta diferentes conjuntos de parâmetros, os quais nem sempre atendem às necessidades de cada usuário. Além disso, não é comum encontrar uma única ferramenta que permita coletar e analisar aspectos de desempenho e energia sobre a execução de uma aplicação, e as que coletam informações sobre o consumo de energia, geralmente não apresentam módulos de visualização para análise dos dados.

Sendo assim, coletar todos os parâmetros necessários é uma tarefa desafiadora, pois requer a utilização das ferramentas de monitoramento de desempenho e energia, as quais possuem diferentes abordagens e técnicas para o registro de informações comportamentais. O estudo das diferentes ferramentas, apresentando a visão geral e os passos para a instalação, com as dependências necessárias, e uso de cada uma delas, pode diminuir a complexidade na sua utilização, permitindo ao pesquisador se concentrar naquilo que é essencial, a análise dos resultados, ao invés de dispender de um longo tempo instalando e configurando as ferramentas que deseja utilizar com suas respectivas dependências. Porém, após a instalação, um outro aspecto percebido é a dificuldade de interpretar os resultados do monitoramento de algumas ferramentas, as quais adotam suas próprias nomenclaturas para os parâmetros.

Além das ferramentas de monitoramento de desempenho abordadas neste relatório, o Nagios também foi estudado. Os passos para sua instalação e uso estão disponíveis no trabalho de (Klôh et al. 2016). Além disso, também foram estudadas outras ferramentas de monitoramento de energia, tais como PowerScope (Noureddine et al. 2013), PowerPack (Ge et al. 2009) e Jalen (Noureddine et al. 2012b). Porém, como elas necessitam de equipamentos de medição externa, os quais precisariam ser adquiridos, não foi possível apresentar os passos para a instalação e o uso.

Referências

[Adhianto et al. 2010] Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., and Tallent, N. R. (2010). Hpctoolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 22(6):685–701. 11

[Bell et al. 2003] Bell, R., Malony, A. D., and Shende, S. (2003). Paraprof: A portable, extensible, and scalable tool for parallel performance profile analysis. In *European Conference on Parallel Processing*, pages 17–26. Springer. 9

[Bourdon et al. 2013] Bourdon, A., Noureddine, A., Rouvoy, R., and Seinturier, L. (2013). Powerapi: A software library to monitor the energy consumed at the process-level. *ERCIM News*, (92). 15

[Browne et al. 2000] Browne, S., Dongarra, J., Garner, N., London, K., and Mucci, P. (2000). A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 42–42. IEEE. 10, 11

[BSC 2017a] BSC (2017a). Barcelona supercomputing center. Acessado em agosto de 2016. 2

[BSC 2017b] BSC (2017b). Tool structure: extracting information from records. 2

[Dongarra et al. 2001] Dongarra, J., London, K., Moore, S., Mucci, P., and Terpstra, D. (2001). Using papi for hardware performance monitoring on linux systems. In *Conference on Linux Clusters: The HPC Revolution*, volume 5. Linux Clusters Institute. 10

[Fernandes 2003] Fernandes, C. A. C. (2003). Estudos de algumas ferramentas de coleta e visualização de dados e desempenho de aplicações paralelas no ambiente mpi. 10

[Ge et al. 2009] Ge, R., Li, D., Chang, H.-C., Cameron, K. W., Feng, X., and Song, S. (2009). Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel Distributed Systems*, 21:658–671. 19

[Hähnel et al. 2012] Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using rapl. *ACM SIGMETRICS Performance Evaluation Review*, 40(3):13–17. 17

[HPCToolkit 2016] HPCToolkit (2016). Hpctoolkit. Acessado em setembro de 2016. 11

[Huck and Malony 2005] Huck, K. A. and Malony, A. D. (2005). Perfexplorer: A performance data mining framework for large-scale parallel computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 41. IEEE Computer Society. 9

[Intel 2015] Intel (2015). Getting started with intel® vtune™ amplifier 2017 for systems. Acessado em setembro de 2016. 3

[Klôh et al. 2016] Klôh, V. P., Ferro, M., Silva, G. D., and Schulze, B. (2016). Performance monitoring using nagios core. Relatórios de Pesquisa e Desenvolvimento do LNCC 03/2016, Laboratório Nacional de Computação Científica, Petropolis - RJ. 19

[McCraw et al. 2014] McCraw, H., Ralph, J., Danalis, A., and Dongarra, J. (2014). Power monitoring with papi for extreme scale architectures and dataflow-based programming models. In *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*, pages 385–391. IEEE. 16

[Mellor-Crummey et al. 2012] Mellor-Crummey, J., Adhianto, L., Fagan, M., Krentel, M., and Tallent, N. (2012). Hpctoolkit user's manual. 11

[Nagel et al. 1996] Nagel, W. E., Arnold, A., Weber, M., Hoppe, H.-C., and Solchenbach, K. (1996). Vampir: Visualization and analysis of mpi resources. 9

[Noureddine et al. 2012a] Noureddine, A., Bourdon, A., Rouvoy, R., and Seinturier, L. (2012a). A preliminary study of the impact of software engineering on greenit. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 21–27. IEEE. 15

[Noureddine et al. 2012b] Noureddine, A., Bourdon, A., Rouvoy, R., and Seinturier, L. (2012b). Runtime monitoring of software energy hotspots. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 160–169. ACM. 15, 19

[Noureddine et al. 2013] Noureddine, A., Rouvoy, R., and Seinturier, L. (2013). A review of energy measurement approaches. *ACM SIGOPS Operating Systems Review*, 47(3):42–49. 19

[NVIDIA 2012] NVIDIA (2012). *NVML API Reference Manual*. `http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf`. 17

[NVIDIA 2017] NVIDIA (2017). Nvidia system management interfaces. Acessado em julho de 2017. 17

[Papi 2016] Papi (2016). Performance application programming interface. Acessado em julho de 2016. 10

[Pillet et al. 1995] Pillet, V., Labarta, J., Cortes, T., and Girona, S. (1995). Paraver: A tool to visualize and analyze parallel code. In *Proceedings of WoTUG-18: transputer and occam developments*, volume 44, pages 17–31. IOS Press. 2

[PowerAPI 2016] PowerAPI (2016). Powerapi. Acessado em outubro de 2016. 15

[RAPL 2016] RAPL (2016). Accessing the intel rapl registers. Acessado em julho de 2016. 16

[Reed 1994] Reed, D. A. (1994). Experimental analysis of parallel systems: techniques and open problems. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 25–51. Springer. 7

[Reed et al. 1992] Reed, D. A., Aydt, R. A., Madhyastha, T. M., Noe, R. J., Shields, K. A., and Schwartz, B. W. (1992). An overview of the pablo performance analysis environment. *Department of Computer Science, University of Illinois*, 1304. 6

[Reed et al. 1993] Reed, D. A., Roth, P. C., Aydt, R. A., Shields, K. A., Tavera, L. F., Noe, R. J., and Schwartz, B. W. (1993). Scalable performance analysis: The pablo performance analysis environment. In *Scalable Parallel Libraries Conference, 1993., Proceedings of the*, pages 104–113. IEEE. 5, 6

[Rodinia 2015] Rodinia (2015). Rodinia:accelerating compute-intensive applications with accelerators. Acessado em junho de 2016. 16

[Shende and Malony 2006] Shende, S. S. and Malony, A. D. (2006). The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311. 8

[Silva et al. 2017] Silva, G. D., Ferro, M., Oliveira, V. D., Klôh, V. P., Yokoyama, A., and Schulze, B. (2017). Estudo de abordagens de monitoramento de desempenho e energia para computação científica. Zenodo. 16

[SINAPAD 2014] SINAPAD (2014). Sdumont, sistema de computação petaflópica do sinapad. Acessado em junho de 2016. 17

[VI-HPS 2017a] VI-HPS (2017a). Virtual institute – high productivity supercomputing. Acessado em agosto de 2017. 2

[VI-HPS 2017b] VI-HPS (2017b). Virtual institute – high productivity supercomputing. Acessado em agosto de 2017. 2

[Weaver et al. 2013] Weaver, V. M., Terpstra, D., McCraw, H., Johnson, M., Kasichayanula, K., Ralph, J., Nelson, J., Mucci, P., Mohan, T., and Moore, S. (2013). Papi 5: Measuring power, energy, and the cloud. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 124–125. IEEE. 16

[Zaki et al. 1999] Zaki, O., Lusk, E., Gropp, W., and Swider, D. (1999). Toward scalable performance visualization with jumpshot. *The International Journal of High Performance Computing Applications*, 13(3):277–288. 9