Otimização de desempenho com o Intel VTune

Sob a lupa

Os processadores de hoje em dia reúnem diversas informações sobre como executar comandos de máquina. Isso permite que os desenvolvedores adaptem seus programas ao hardware da melhor forma possível, o que contribui de forma importante à aceleração de sua execução. Para realizar essa tarefa em processadores Intel, temos o auxílio do VTune.

por Michael Hebenstreit

CPUs modernas possuem uma série de formas para acelerarmos programas. Isso significa, por outro lado, que o número de armadilhas ocultadas por um desempenho otimizado também é maior. Normalmente podemos demonstrar como um programa reage à carga apresentada de acordo com o tempo. Com isso, a otimização do software tem uma importância cada vez maior. O desenvolvedor precisa sempre, no entanto, adaptar-se aos hardwares disponíveis e suas possibilidades. O *Intel Vtune Performance Analyser* [1] facilita a localização desses problemas.

Antes de um programa ser executado, ele é traduzido do código-fonte para a linguagem da máquina, em várias etapas. Não há diferença se esse processo foi executado pelo compilador no desenvolvimento do programa ou em tempo de execução pelo interpretador – ao final, a CPU ainda vê apenas o código de montagem binário codificado.

Cada instrução consiste de um ou mais códigos de bits, que a CPU entende como uma operação elementar. Por exemplo, baixar o dado do endereço 0x1234 para o registro *RAX*. Internamente, uma CPU moderna não processa uma instrução desse tipo diretamente, mas dividida em pequenas micro-operações – chamadas *µops*. Esse procedimento também faz com que o comando percorra, no mínimo, os seguintes níveis:

- Baixar um comando;
- Decompor o comando em µops;
- Executar as µops e calcular os endereços necessários;
- (Opcional) Baixar os dados da memória;
- Escrever o resultado na memória.

Em arquiteturas reais de processador, a subdivisão continua de modo que no final são necessários, normalmente, dez ou mais passos – em alguns sistemas são mais de 30. Em vez de executar um comando de montagem após o outro, assim que um comando passa do primeiro nível, o comando seguinte migra para a "linha de montagem", chamada de *pipeline*.

Sob condições adequadas, esse processo pode permitir que códigos puramente seriais sejam executados com o máximo de desempenho. Entretanto, na prática isso raramente é possível; sempre ocorrem imprevistos devido a ramificações e laços no programa. Em vez de simplesmente esperar para saber qual será o resultado referente a uma ramificação, o prognóstico da lógica da CPU tenta reconhecer o caminho mais provável.

Se ao final da pipeline o sistema reconhecer que a previsão estava errada, ele interrompe todas as funções ativas na pipeline no momento, carregando-a novamente. Esse evento é chamado de *branch misprediction*, e custa naturalmente muito tempo.

Cache

Outra modalidade de redução de desempenho bastante recorrente é o acesso da CPU à memória principal. CPUs atuais são operadas a freqüências de *clock* de dois a quatro GHz; ou seja, um clock dura meio nanossegundo. Caso a CPU tenha que acessar a memória principal, existem tempos de espera de 70 a 100 ns, ou seja, de até 600 clocks. Durante esse tempo o processador geralmente é incapaz de desempenhar qualquer trabalho significativo. Essa problemática deve ser combatida pelos *caches*, que minimizam o tempo de acesso através de um armazenamento mais rápido. Hoje em dia, eles são consideravelmente menores que os programas reais ou registros de dados. O programa também deve estar configurado de tal maneira que o cache seja utilizado da forma mais eficiente.

Além desses dois exemplos, existem muitas outras possibilidades de otimização. Felizmente os hardwares modernos ajudam o programador nessa função. Os processadores baseados no *Intel Pentium* ou *Core 2 Duo* conseguem registrar a ocorrência dos eventos. Em cada processador existem alguns registros de números, cujo valor é incrementado a cada ocorrência do evento mensurado. Caso um numerador ultrapasse um limite predeterminado – descrito na tecnologia Intel como SAV (*Sampling After Value*) –, a CPU provoca uma interrupção. Softwares específicos, isto é, drivers do sistema operacional, podem reagir à interrupção e interpretar o evento.

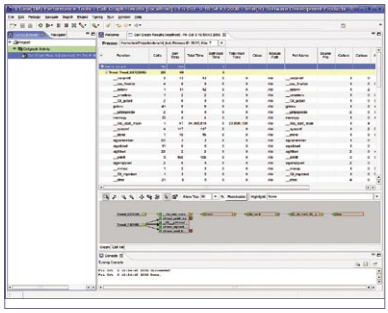

Figura 1: O primeiro resultado da análise do *Call Graph* (acima em tabela, abaixo disponibilizada em gráfico). O diagrama mostra os anexos lógicos, sem considerar os números.

Verificação de problemas na CPU

Os conceitos de *Event* e *Counter* já foram mencionados várias vezes sem que tenhamos explicado seu significado prático. Exemplos de dois eventos freqüentemente utilizados em sistemas Pentium são *Clockticks* e *Instructions retired*. Os primeiros medem o tempo e são incrementados a cada sinal do clock interno (como por exemplo cada nanossegundo). O último representa o número de instruções executadas.

A relação entre os clockticks e as Instructions retired (também chamadas de *CPI*, *Clocks per Instruction*) mostra quanto tempo uma instrução demorou para ser executada. Como as CPUs modernas executam paralelamente mais de uma instrução, esse número pode ser menor que 1. Na prática, isso raramente é alcançado, como por exemplo na compressão e descompressão de vídeo com comandos *SSE*. Valores entre 1 e 1,5 são bons, enquanto aqueles superiores a 4 devem ser aperfeiçoados.

A medida de clockticks de um programa, uma função ou até mesmo de linhas de código individuais mostra ao desenvolvedor de quanto tempo uma parte do programa necessita para a execução. O valor de CPI informa se a execução foi eficiente do ponto de vista da arquitetura da CPU.

Um meio auxiliar para registrar e interpretar esses eventos é o programa desenvolvido para processadores Intel, o Vtune. Ele não pode ser utilizado em outros produtos compatíveis com x86, pois o contador de hardware necessário não faz parte da definição da arquitetura x86, sendo implementado de maneira diferente por outros fabricantes, como a AMD. Cada fabricante de hardware compatível com x86 acaba sendo obrigado a desenvolver sua própria microarquitetura. Como o contador de eventos está profundamente vinculado à microarquitetura, a compatibilidade entre os fabricantes não é possível. Apesar da diferença entre arquiteturas, a regra mostra que o Vtune encontra gargalos de desempenho em todos os fabricantes.

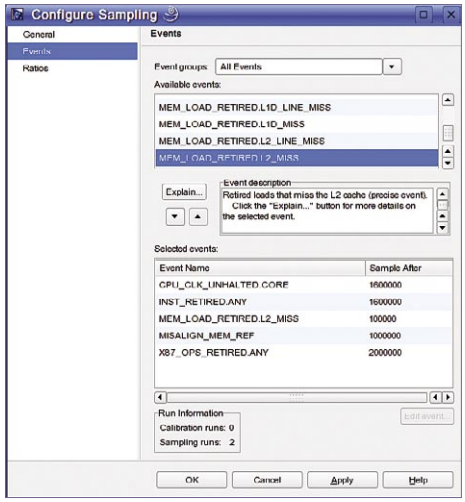

Figura 2: O Vtune permite selecionar os próprios eventos, além dos pré-instalados.

Contar eventos

Não faria sentido registrar imediatamente cada ocorrência de evento e determinar em qual seção do código ele se deu. Se tentássemos fazer isso com o evento Instruction retired, por exemplo, o tempo de execução de um programa aumentaria múltiplas vezes. Então, em vez disso é utilizado um método estatístico. O contador é calculado através do valor SAV (Sample After Value), escolhido de forma a fazer com que a CPU pare o programa através de uma interrupção depois de aproximadamente um milissegundo. O sistema de tempo de execução do Vtune salva no gerenciador de interrupções a informação do comando em execução (*Program Counter Register*) e o número da *thread* sendo executada, assim como o do processo. Esses dados são extraídos pelo Vtune, na avaliação da medição.

Em um processador de 2 GHz, isso significa que todos os dois milhões de ciclos de clock teriam determinado em que posição o programa se encontrava imediatamente. Se cada linha do programa rodar exatamente uma vez, a subdivisão deveria ser equivalente. Mas *jumps*, laços e ramificações produzem os chamados *hotspots*, ou seja, partes de programa que absorvem uma parte considerável do tempo de execução. Além disso, o desempenho de programas pode

ser degradado pelo aumento de tempos de latência no acesso à memória, ao disco ou à rede.

Ao se realizar uma medição, deve-se também levar em consideração que o Vtune utiliza um método estático para avaliação de um programa para os hotspots. Os seguintes pontos são importantes:

- O programa deve trabalhar com dados realistas (isso também vale naturalmente para qualquer outro tipo de análise de desempenho).
- O tempo de execução deve ser longo o suficiente para que apareçam, de maneira estática, eventos suficientes em cada hotspot. O Vtune ainda pode apresentar o número de eventos contabilizados de fato. Caso a medição de um hotspot dependa de apenas poucos eventos (menos de 20, por exemplo), a informação é menos confiável.
- Teoricamente pode ocorrer de os hotspots não serem conhecidos.
- É freqüente a medição apresentar uma imprecisão – chamada *skid* –, na qual o evento real e a linha de programa mostrada não combinam exatamente. Nesse ponto é necessária uma pequena experiência, mas deve estar claro para qualquer programador que, por exemplo, apenas no caso de uma bifurcação pode ocorrer um Branch Mispredict.

Além dessas limitações, há algumas vantagens:

- Não é necessário orquestrar o próprio código-fonte.
- O Vtune permite uma visão de todo o sistema. O autor conseguiu várias vezes, por exemplo, com a ajuda do Vtune, reverter problemas de desempenho em um processo complicado que não tinha nada a ver com o próprio aplicativo.

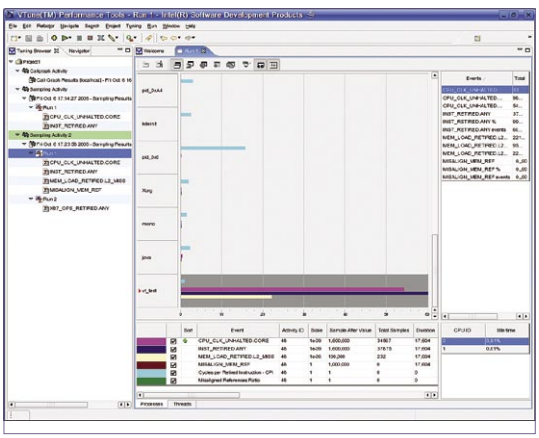

Figura 3: O resultado de um *Sampling Run* mostra os eventos estáticos de todos os processos que estão rodando no sistema. O objeto da pesquisa é mostrado na linha cinza na parte de baixo.

Primeiros passos

Na maior parte dos novos sistemas Linux, a instalação não apresenta problemas. Normalmente as versões atuais do Suse, Red Hat e Fedora encontram bom suporte e também são certificadas. A versão atual do Vtune também pode ser utilizada sem problemas no Open Suse 10.1 (32 bits), embora ela ainda não suporte essa variante oficialmente.

A instalação requer o arquivo compactado e um arquivo de licença válido. Em [1] há também uma licença de teste de 30 dias. A versão econômica para estudantes custa exatos 50 euros, enquanto uma única licença normal custa a partir de 630 euros. O processo de instalação é iniciado pelo script `install.sh`. Aconselha-se, inicialmente, a manter a configuração padrão.

Módulo de kernel obrigatório

O Vtune necessita de um diretório de dados globais, que ele por padrão deposita em `/opt/intel/vtune/global_data`. Ele requer um módulo de kernel próprio, cujo código-fonte encontra-se em `/opt/intel/vtune/vdk/src`. Em novas versões do kernel e de distribuições, isso pode levar a problemas que, com certa sorte, normalmente são eliminados.

O Vtune dispõe também de uma interface gráfica de usuário (*vtlec*), bem como de uma ferramenta para linha de comando (*vtl*). No Linux, a interface gráfica se baseia na plataforma do *Eclipse*, e apresenta as solicitações feitas à CPU e à memória. No entanto, em alguns casos essas solicitações podem causar adulterações à medição da capacidade. Por isso o autor aconselha a separar a medição da análise.

Interface gráfica

Como exemplo prático, em [2] há um pequeno programa em *C*, com o qual os testes a seguir foram executados. Ele se baseia em *OpenMP* e com isso roda paralelamente em várias plataformas. A tradução foi feita com o compilador C da Intel, com o comando `icc -g -o vt_test -openmp omp_mm4.c`. A opção `-g` é importante; a interpretação da medição exige as informações dos símbolos, caso contrário o Vtune apenas consegue mostrar uma classificação das linhas de Assembler, exatamente como um depurador.

A parte gráfica da aplicação é iniciada com `/opt/intel/vtune/bin/vtlec`. O menu *File* | *New* | *New Project* | *Tuning Activity* inicia uma nova análise. A primeira verificação normalmente diz respeito ao chamado *Callgraph*. O Vtune pesquisa o controle de fluxo no nível das chamadas de função, e atribui a elas eventos medidos. Através da instrumentação do código a ser executado, obtemos dados absolutamente corretos, como, por exemplo, com que freqüência o programa invocou uma dada função, e que parte do tempo de execução ele passou lá. Assim, o programador visualiza imediatamente quais funções contribuem com uma parte significativa do tempo de execução como um todo.

Um assistente auxilia todas as configurações necessárias. Normalmente basta inserir o caminho para a aplicação com o parâmetro de inicialização necessário. O Vtune carrega o binário (para examinar os pontos de entrada e saída de funções) e executa o programa. Depois de terminar a execução do programa, ele analisa os dados e os mostra sob as formas de tabela e Callgraph (**figura 1**).

Do lado esquerdo o programa mostra uma estrutura em árvore das medições. Um duplo clique em uma pasta *Run* (ou em uma das medições listadas abaixo) abre a visão de dados. Ela é mostrada pelo Vtune através do menu contextual (clique direito do mouse) como tabela e gráfico de barras.

Análises precisas requerem uma rodada de coleta de informações, chamada *Sampling Run*, que inicia o *Sampling Wizard* na *Tuning Activity*. Caso o usuário queira ir além da instalação padrão e ainda medir eventos, ele precisa marcar com um `x` a opção *Change Sampling Events*, na caixa de diálogo do assistente. Em *Activity Configuration*, é possível inserir medições adicionais através de *Events* (**figura 2**).

O número de eventos medidos ao mesmo tempo em uma execução depende da CPU e da combinação dos próprios eventos. O Vtune gera automaticamente um número de execuções de acordo com a definição da medição, realizando-as em seguida. Isso naturalmente significa também que cada rodada executa o programa duas vezes: uma vez para a calibragem, outra para a medição. O Vtune presume

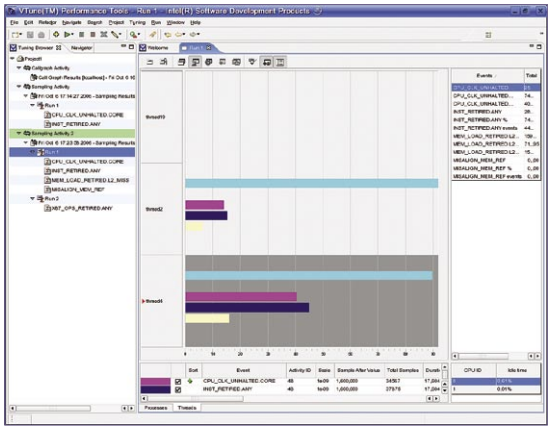

Figura 4: A visualização da *thread* denuncia: uma thread está recebendo mais tarefas do que outra, e por isso também dura mais tempo. Além do tempo de execução, podem ser vistas as respectivas partes dos eventos registrados pelo programa.

então que o programa trabalhe de forma idêntica todas as vezes.

Eventos e contador

Em CPUs *Core 2* existem cinco contadores, em princípio. Deles, três estão limitados aos eventos padrão, restando apenas dois contadores para medições em geral. Entretanto, na maioria das vezes faz sentido definir os eventos padrão para cada caso. Com isso, comportamentos como `INST_RETIRED.ANY/CPU_CLK_UNHALTED.CORE` sempre podem ser avaliados. A lista dos eventos disponíveis depende da CPU, e em sistemas mais modernos também do chipset. Eventos como "2nd Level Cache read Misses" podem ser encontrados em todas as CPUs atuais, mas contadores SSE3, por exemplo, estão presentes apenas em sistemas com processadores que também implementam essa unidade.

É importante lembrar que a Intel não implementou o contador de forma sistemática e abrangente para todos os processadores. Os processadores *Pentium 4*, *Xeon* e *Pentium M* são relativamente semelhantes entre si, distanciando-se, no entanto, da arquitetura completamente oposta do Itanium. O novo *Core 2* se orienta em torno de ambos os antecessores, e é equipado com mais de 700 eventos, também estruturados sistematicamente.

O resultado de um Sampling Run é mostrado primeiramente englobando todos os processos (**figura 3**) que rodaram no sistema durante o tempo de medição. Se o cursor do mouse ficar parado por alguns instantes sobre o nome do módulo, o Vtune mostra um resumo desse módulo.

No exemplo, o programa `vt_tes` teve o maior valor, bem como o número de instruções executadas (`INST_RETIRED.ANY`), e também o tempo de execução (`CPUT_CLK_UNHALTED.CORE`). O último critério, à primeira vista, é o mais importante. Otimizações se pagam nos pontos onde um tempo de execução mais longo é necessário. Nos hotspots, a relação do número de instruções fornece uma base para nos informar se uma otimização vale a pena do ponto de vista da microestrutura.

Primeira avaliação

Em tais decisões deve-se sempre observar quanto tempo realmente seria necessário para um segmento de código. Em um programa que rode em dez segundos, não é interessante otimizar uma parte de código com um valor de CPI de 20, pois ele necessita de apenas 0,1 segundo para a execução. Também é importante o fato de que não se atinge um ganho de velocidade se mais de uma instrução for executada em um mesmo ciclo do clock. Por isso, é interessante verificar antes se o algoritmo pode ser aperfeiçoado ou se a paralelização não seria um caminho possível e mais promissor.

No exemplo, esse definitivamente não é o caso. Um duplo clique na entrada do `vt_test` abre a visualização de threads (**figura 4**). Logo se percebe que as diferentes threads OpenMP não recebem a mesma quantidade de trabalho. Elas executam um número bem diferente de instruções e, obviamente, também precisam de períodos diferentes. Até mesmo o comportamento de cada thread individual está longe do ideal.

Exemplo 1: Laço principal

```
01 for (k = 0; k < REPETIÇÕES; k++) {
02    #pragma omp for
03    for(i = 0; i < RA; i++) {
04       for(j = 0; j < i; j++) {
05          d[i] += a[i][j] * b[i] + c[i];
06       }
07    }
08 }
```

Processo de otimização

Outro duplo clique em uma das threads abre a visualização de módulo. O usuário pode navegar com o mouse através dos hotspots e chegar até o código-fonte (**figura 5**), cujo laço interno pode ser visto no **exemplo 1**. Cada laço realiza em princípio o mesmo trabalho – a diferença é o tipo de armazenamento de dados.

A diferença entre o segundo e o terceiro laços está no acesso à matriz `a[][]`. O último – e pior – caso endereça o primeiro índice, portanto de forma contrária à estrutura de caching e pre-fetching do hardware. Cada erro de acesso ao cache (*Cache Miss*) baixa não apenas os dados necessários, mas também os elementos relativos a eles para o cache. Os laços 1 e 2 também reutilizam essas datas imediatamente. O terceiro requer um campo de memória completamente diferente a cada passo, o que leva a um grande aumento de cache misses. Isso pode ser observado não apenas no Vtune, mas também no tempo de execução (**exemplo 2**).

A diferença de peso entre ambas as threads é esclarecida por um erro comum na aplicação do OpenMP. A thread 1 executa o laço de 0 até $N/2$, e a thread 2, de $N/2$ até N. A duração dos laços internos aumenta de forma proporcional a N. Esse é o motivo pelo qual o tempo de execução na utilização de duas CPUs não se divide ao meio. O leitor que se interessar deve buscar a especificação do OpenMP [3], seção *Schedule*.

Caso o `vtlec` realize a medição, pode acontecer de esse processo con-

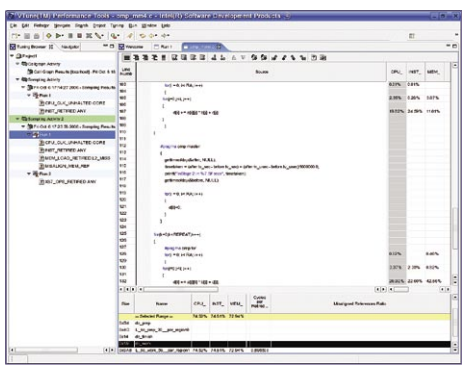

Figura 5: A visualização do código-fonte quebra os dados reunidos até a única linha de código para baixo. Na coluna cinza à direita, a parte da CPU pode ser vista por qualquer aplicação.

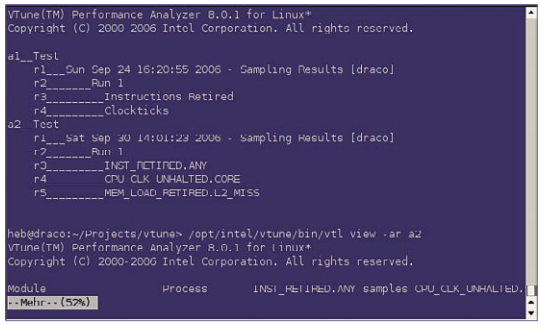

Figura 6: Informações importantes para uma otimização são fornecidas também pela ferramenta de linha de comando `vtl`. No exemplo, é mostrada uma visualização dos eventos registrados.

sumir grande parte da capacidade de CPU disponível, principalmente em conjunto com o *X11* e o gerenciador de janelas. Além disso, em tarefas em lote, operações interativas costumam ser impossíveis. Por isso, é interessante que grande parte da funcionalidade do Vtune também possa ser acessada pela linha de comando (**figura 6**). Nesse caso, `vtl -help` fornece uma visão geral dos comandos e opções disponíveis. A ajuda oferece informações detalhadas quando informamos o comando desejado como opção adicional. Depois é conferida uma posição especial do comando padrão para a medição: `vtl -help -c sampling`.

Linha de comando do Vtune

O Vtune fornece não somente os detalhes do comando desejado, mas também uma lista de todos os eventos disponíveis. Sem necessitar de mais carregamentos, o programa de teste pode ser executado no Vtune como a partir do `vtlec`:

Exemplo 2: Threads

```
01 $ export OMP_NUM_THREADS=2
02 $ ./vt_test
03 example with 2 threads
04
05 Stage 1 in 5.06335 secs
06 Stage 2 in 5.05060 secs
07 Stage 3 in 6.87645 secs
08 Finished
09
10 $ export OMP_NUM_THREADS=1
11 $ ./vt_test
12 example with 1 threads
13
14 Stage 1 in 6.82961 secs
15 Stage 2 in 6.70549 secs
16 Stage 3 in 9.10998 secs
17 Finished
```

```
vtl activity Test -d 40 -c sampling -o "-
➥ec en='Instructions Retired'
➥en='Clockticks' -cal yes" -app ./vt_test
➥run
```

O elemento `activity Test` define uma nova medição, que com `-d 40` deve durar no máximo 40 segundos (isso evita problemas com programas dependentes). É conduzida uma medição do tipo *sampling*, que analisa dois eventos. Para calibrar o contador, o Vtune executa automaticamente, com a opção `-cal yes`, duas funções: a primeira simplesmente conta todos os eventos que ocorrerem, enquanto a segunda conduz a medição propriamente dita.

O comando anexo `run` inicia a medição definida da mesma forma. Caso contrário, o comando acima iria definir apenas uma medição, que o desenvolvedor iniciaria em um momento mais adiante através de `run`. Os dados são depositados pelo Vtune no diretório pessoal em `~/VTune`. Na linha de comando, `show` mostra os dados salvos atualmente e, com a opção `all`, são mostrados também os eventos salvos em uma execução: `vrt show -all`.

Dados de teste transportáveis

É possível ter uma primeira visão geral da medição no console com o comando `vtl view test -modules`. Com isso o Vtune mostra todos os processos que estavam ativos durante a medição, assim como os valores medidos. As outras avaliações infelizmente precisam de um pouco mais de trabalho, sendo produzidas no console. Os dados coletados podem ser empacotados e transferidos para outro computador para avaliação. No primeiro passo, `vtl pack test.vxp -ar test` empacota os resultados no arquivo do pacote `rtest.vxp`. Ele contém, além do próprio arquivo de dados (`*.tb5`), informações sobre que dados foram medidos, e quando. Os dados salvos há pouco podem ser lidos na interface gráfica `vtlec` por meio de *File* | *Import* | *Tuning File* | *Export File* (`*.vxp`).

Além das análises apresentadas, o Vtune também oferece a possibilidade de conduzir avaliações em um espaço de tempo determinado, o que é de grande ajuda para localizar gargalos no decorrer de eventos.

Para facilitar o desenvolvimento de programas paralelizados, o *Intel Thread Checker* e o *Intel Thread Profiler* trazem duas extensões especiais para o Vtune. O Thread Checker testa programas baseados em OpenMP, *Posix* Threads ou Windows® Threads em sua correção, e sobretudo se os acessos à memória estão programados corretamente e não há possibilidade de ocorrerem condições de corrida.

O Thread Profiler facilita a verificação e confirmação do comportamento temporal de um programa baseado em threads, registrando quais threads iniciadas estão ativas em cada momento. Isso propicia também a localização de *deadlocks*, ocorrências tão indesejáveis em qualquer programa.

Resultado

O Intel Vtune ajuda os programadores a localizarem problemas decorrentes de um mal ajuste do programa ao hardware Intel. O autor pôde verificar na prática que esses problemas, em sua maioria, ocorrem com menor ou maior força em sistemas compatíveis com x86; portanto, uma possível otimização é indicada a todos os sistemas desse tipo.

O Vtune também pode fornecer diversas informações ao iniciante. Não é necessário escrever ou carregar códigos de forma especial. Muitos problemas podem ser compreendidos sem acesso ao código-fonte, já que o usuário obtém uma imagem completa do sistema com uma medição.

Com isso, o Vtune se revela uma ferramenta com muitas possibilidades, que além disso é fácil de ser instalada, mas que requer um certo conhecimento do hardware utilizado. A capacidade de reconhecer quais dos muitos eventos mensuráveis são relevantes ao se estudar uma aplicação é uma das tarefas que nenhum programa pode resolver de forma autônoma. ■

Mais Informações

[1] Intel Vtune: http://www.intel.com/cd/software/products/asmo-na/eng/vtune/239144.htm

[2] Exemplos on-line: http://www.linuxmagazine.com.br/issue/27/vtune.html

[3] OpenMP: http://www.openmp.org