Um Estudo da Adição de um Quarto Nível de Cache em Processadores Multicore

Carlos Eduardo B. Bezerra[1], Cláudio F. R. Geyer[2]
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre, RS, Brasil
{[1]carlos.bezerra, [2]geyer}@inf.ufrgs.br

Resumo

Após uma certa estagnação do aumento da freqüência dos processadores, tem-se investido em arquiteturas multicore *(com dois ou mais núcleos) para suprir a demanda por capacidade de processamento imposta pelas aplicações cada vez mais pesadas. Estas aplicações também têm uma exigência cada vez maior de espaço de armazenamento em memória principal, o que exige um proporcional aumento da memória cache dos processadores. Contudo, a pura e simples adição de memória cache a um processador pode não apenas não aumentar sua performance, como também diminuir sua eficiência, ao aumentar o tempo de acesso a uma memória muito grande. Neste trabalho, é feita uma comparação entre o desempenho de uma arquitetura multicore com uma cache L3 grande e uma arquitetura com um nível adicional, L4, de memória. Além disso, é proposta uma arquitetura com hierarquia de cache entrelaçada, com o objetivo de diminuir a profundidade de busca na memória quando dois núcleos compartilham uma variável. Verificou-se que uma memória com quatro níveis melhor estruturada executou as tarefas simuladas em menos tempo do que uma arquitetura com uma grande cache L3. Por fim, evidenciou-se que o entrelaçamento da hierarquia de cache pode trazer benefícios significativos.*

1. Introdução

Nos últimos anos, devido a uma certa estagnação da freqüência de relógio dos processadores [7], os fabricantes têm optado pela produção de processadores com vários núcleos de processamento (processadores *multicore*, com diversos *cores*). Exemplos disso são as arquitetura da Intel (*Dunnington*, *Nehalem* etc.), AMD (*Barcelona*, *Budapest*, *Deneb* etc.) e Sun (*Niagara*, *Victoria Falls* etc.).

As memórias cache servem para reduzir o número de ciclos de latência que um processador deve esperar para acessar um determinado dado na memória. Reduzindo o número de ciclos, reduz-se o tempo necessário para acessar um dado. Por esta razão, o tempo de acesso a dados na memória cache é consideravelmente menor do que o tempo necessário para acessar a memória principal, por exemplo.

Contudo, ultimamente as aplicações têm cada vez um volume maior de dados a processar. Essa quantidade maior de dados, se não puderem ser acessados rapidamente pela unidade de processamento, podem implicar um pior desempenho do sistema [5]. Outra questão é que, nas arquiteturas multicore, vários núcleos podem estar utilizando a memória ao mesmo tempo. Por causa dessas razões, é recomendável aumentar o tamanho da memória cache (que tem acesso rápido) proporcionalmente ao aumento do volume de dados das aplicações e ao número de núcleos do processador.

A grande questão é que aumentar a memória cache, pura e simplesmente, pode não ser suficiente para reduzir o tempo de execução das tarefas realizadas pelo processador. Na verdade, um aumento sem critérios do espaço da memória cache pode implicar uma redução da velocidade do sistema. Isso acontece porque quanto maior a memória, maior é o seu tempo de acesso, devido a uma estrutura mais complexa de endereçamento [3]. Além disso, se vários núcleos estão tentando acessar a memória ao mesmo tempo, haverá uma contenção pelo barramento de acesso àquela memória, fazendo com que um ou mais processadores esperem até que o barramento seja liberado. Uma má escolha de estrutura e tamanho de memórias cache pode fazer com que uma arquitetura com mais memória seja mais lenta do que uma arquitetura com menos memória, porém estruturada de maneira mais eficiente.

Tendo esses aspectos em vista, neste trabalho foi feito um breve estudo da influência da estrutura de cache em processadores multicore, comparando o desempenho de uma hierarquia com 3 níveis de memória, comparada com outras duas hierarquias, ambas com 4 níveis, introduzindo

um nível L4 compartilhado entre todos os núcleos. Como contribuição, neste trabalho foi testado o uso de uma hierarquia de memória entrelaçada – o que será descrito nas próximas seções –, com o objetivo de reduzir a profundidade da busca realizada por um núcleo para buscar um dado compartilhado com outro núcleo.

O texto está organizado da seguinte forma: na seção 2, é apresentada em maiores detalhes a proposta deste trabalho, incluindo a idéia de entrelaçar a hierarquia da memória cache; na seção 3, são apresentados os detalhes da modelagem e parâmetros das simulações que foram realizadas; na seção 4, são mostrados os resultados e uma breve análise dos mesmos e, na seção 5, são apresentadas as conclusões a que se chegou com este trabalho.

2. Proposta

Este trabalho teve dois objetivos, que são procurar evidências de que a adição de um quarto nível de memória cache é mais benéfico do que aumentar a memória cache L3 e avaliar o ganho de desempenho conseguido ao se entrelaçar a hierarquia de memórias cache. Para analisar a diferença de desempenho com uma cache L3 grande, contra a adição de um módulo de memória L4, foram comparadas as arquiteturas seguintes:

- uma arquitetura multicore, inspirada na arquitetura Dunnington, utilizada no Intel Xeon, que dispunha de 8 núcleos de processamento, com quatro módulos de memória cache L2, cada um compartilhado por um par de núcleos, e uma grande memória L3 (com 32 megabytes de espaço), compartilhada entre todos os oito núcleos, como mostrado na Figura 1 – nas figuras, *dc* e *ic* referem-se, respectivamente, a cache de dados e cache de instruções, que compõem a cache L1;

- uma arquitetura multicore, semelhante à anterior, também com oito núcleos, porém os 32 megabytes que eram da L3 foram divididos em: duas memórias L3 de 8 megabytes, cada uma compartilhada por metade dos núcleos, e uma memória L4 de 16 megabytes, compartilhada por todos (Figura 2).

Outra contribuição do trabalho foi que também se propôs o entrelaçamento da hierarquia das memórias cache. Para buscar algum resultado que indique se há algum benefício ao ser utilizado esse tipo de abordagem, foi comparadas as seguintes arquiteturas:

- uma arquitetura com quatro níveis de cache (L1, L2, L3 e L4), onde cada módulo de cache só tem um módulo de nível inferior na hierarquia de memória, tal qual na Figura 2;

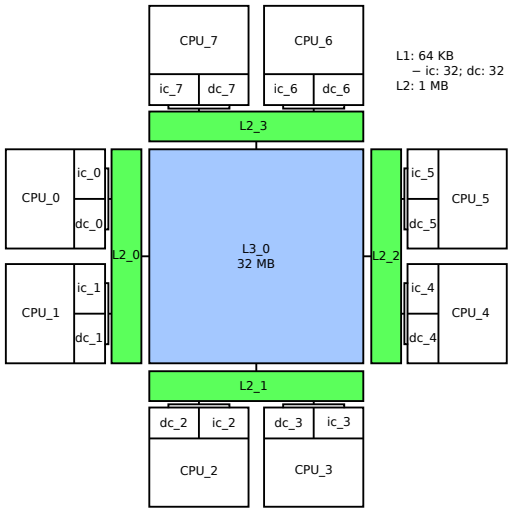

Figura 1. Arquitetura com 3 níveis de memória cache

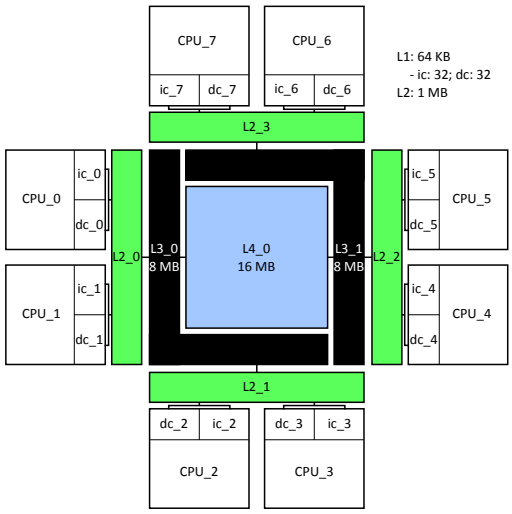

Figura 2. Arquitetura com 4 níveis de memória cache

- com outra arquitetura, na qual cada módulo de cache L2 pode ter uma ou duas memórias L3 no nível inferior da hierarquia. A Figura 3 ilustra essa arquitetura de hierarquia entrelaçada, e a Figura 4 apresenta de maneira mais clara essa diferença das hierarquias – na Figura 4 (a), temos a arquitetura sem entrelaçamento e, na Figura 4 (b), com entrelaçamento.

O objetivo de fazer esse entrelaçamento é tentar reduzir o problema de poder haver buscas muito profundas na hierarquia das memórias cache quando dois núcleos de processamento compartilham uma variável. Seja uma situação em

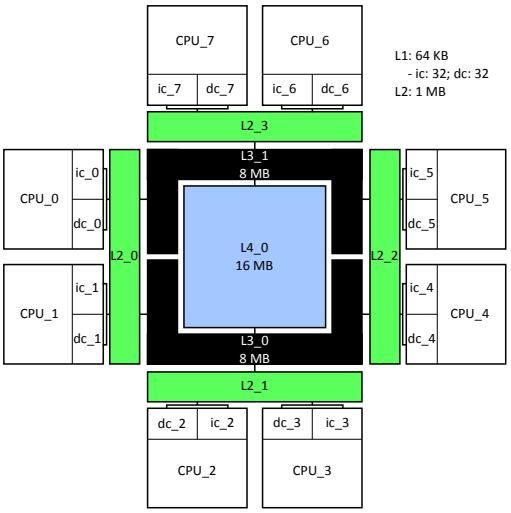

Figura 3. Arquitetura com 4 níveis de memória cache e hierarquia entrelaçada

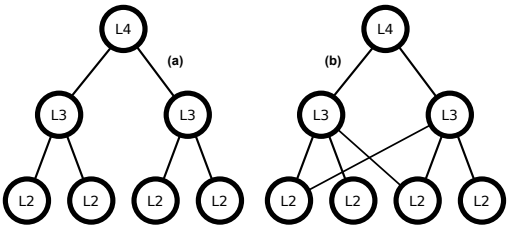

Figura 4. Árvores representando as hierarquias avaliadas

que os núcleos *CPU_0* e *CPU_7* (Figura 2) lêem e escrevem na mesma variável *x*. Sem o entrelaçamento, quando um deles escrevesse nessa variável, o valor possivelmente armazenado na cache L1 do outro estaria incoerente e aquele núcleo teria que fazer uma busca até a primeira área de memória comum entre os dois núcleos – no caso, a memória L4, que é mais distante dos núcleos e teria um tempo de acesso provavelmente maior, reduzindo o desempenho do sistema.

No entanto, considerando a arquitetura com hierarquia entrelaçada, apresentada na Figura 3, bastaria que o dado estivesse atualizado na cache de terceiro nível *L3_1*, que não seria necessário fazer uma busca tão profunda quanto mergulhar na árvore até o nível das caches L4. Utilizando essa abordagem, diminui-se a maior profundidade mínima de busca para que os núcleos compartilhem dados nas memórias cache. Contudo, esse mecanismo tem alguns detalhes a serem considerados numa implementação real da arquitetura. Por exemplo, quando ocorre um *cache miss* em uma cache L2, pode ser necessário fazer busca em duas caches L3 – no exemplo da Figura 3, isso acontece quando ocorre um cache miss nos módulos *L2_0* e *L2_2*. Isso se refletirá em um tempo maior de acesso, se as buscas forem seqüenciais, ou em maior custo de fabricação do hardware, se houver no processador uma unidade responsável por realizar as buscas em paralelo e retornar aquela que encontrar o dado procurado primeiro.

Na seção a seguir, serão apresentados os detalhes da modelagem das arquiteturas propostas e dos testes executados através de simulação.

3. Modelagem e simulações

Para realizar a simulação das arquiteturas propostas na seção 2, foi utilizado o simulador Simics [6], versão 4. Através do Simics, é possível modelar e simular diversas arquiteturas de computadores, sobre as quais podem ser executados sistemas operacionais populares, como o Linux. Para modelar a arquitetura e os componentes de hardware, o Simics utiliza arquivos escritos com uma linguagem de modelagem de dispositivo (DML, *device modeling language*), que permite criar componentes, ajustar seus atributos e conectá-los entre si. Após instalado e configurado o sistema operacional, foi instalado o programa de avaliação de desempenho NAS (*Numerical Aerodynamic Simulation*), que permite a avaliação de desempenho de arquiteturas de computadores através de extensos cálculos numéricos [1].

A modelagem das arquiteturas propostas foi feita com base em uma arquitetura sparc64 (TI UltraSparc II – *BlackBird*). Todas têm as seguintes características em comum:

- Número de núcleos de processamento: 8;
- Freqüência de clock de cada núcleo: 168 MHz;
- Memória cache L1 de dados: 32 kilobytes;
- Memória cache L1 de instruções: 32 kilobytes;
- Latência da cache L1: 2 ciclos;
- Memória cache L2: 1 megabyte (4 módulos de 1 MB, cada um compartilhado por um par de núcleos);
- Latência da cache L2: 5 ciclos.

A partir do terceiro nível de cache, as arquiteturas mudam completamente, sendo que três propostas foram modeladas:

- Arquitetura **L123**, com três níveis de cache:
  - Memória cache L3: 32 megabytes (compartilhados por todos os núcleos);
  - Latência da cache L3: 30 ciclos;

- Arquitetura **L1234**, com quatro níveis de cache (sem entrelaçamento):
  - Memória cache L3: 8 megabytes (2 módulos de 8 MB, cada um compartilhado por quatro núcleos);
  - Latência da cache L3: 15 ciclos;
  - Memória cache L4: 16 megabytes (compartilhados por todos os núcleos);
  - Latência da cache L4: 30 ciclos.
- Arquitetura **L4Interleaved**, com quatro níveis de cache e com entrelaçamento:
  - Memória cache L3: 8 megabytes (2 módulos de 8 MB, cada um compartilhado por quatro núcleos, porém o conjunto de núcleos é diferente da arquitetura L1234, entrelaçando os níveis de cache);
  - Latência da cache L3: 20 ciclos;
  - Memória cache L4: 16 megabytes (compartilhados por todos os núcleos);
  - Latência da cache L4: 30 ciclos.

Os motivos para decidir-se por esses parâmetros foram os seguintes: na arquitetura L123, a cache L3 tem 32 megabytes de espaço, causando um tempo de acesso (30 ciclos) maior do que na arquitetura L1234 (15 ciclos), onde cada cache L3 tem apenas 8 megabytes. Já na L4Interleaved, cada cache miss em uma cache L2 pode precisar de duas buscas nas caches L3. Para simular esse comportamento, definiu-se o tempo de acesso às L3 como de 20 ciclos, como uma estimativa de tempo médio de acesso.

O sistema operacional que foi instalado sobre cada uma das arquiteturas simuladas foi o Ubuntu (GNU/Linux, kernel 2.6.15-53). O software utilizado para avaliação das arquiteturas modeladadas, o NAS, consiste de diversos programas de avaliação de desempenho. Dois deles são o **BT** e o **CG**. Detalhes a respeito do que fazem podem ser encontrados em [1] e [4].

Para cada combinação testada (arquitetura $\times$ programa), foram executadas três repetições, devido a restrições no tempo que se tinha disponível para realizar os testes. Na média cada repetição da simulação demorava três horas. Estima-se que, apenas executando três vezes cada simulação, tenha-se dispendido 54 horas. Na próxima seção, serão apresentados os resultados encontrados com as simulações que foram realizadas.

4. Resultados

Tanto o CG quando o BT buscam estressar os componentes de processamento e memória. O BT tem um tempo de execução maior, permitindo a avaliação do processador, enquanto que o CG visa a estressar as memórias, permitindo o teste de desempenho da estrutura de cache simulada. Foram utilizadas implementações em OpenMP [2] de ambos. Para coletar os dados, foi utilizada a *magic instruction* do Simics, que permite pausar a simulação e extrair o estado de cada um dos componentes de hardware. Os valores extraídos com esse método, para cada programa executado (BT e CG) foram:

- Número de ciclos do processador;
- Número de instruções executadas pelo processador – no Simics, é utilizado para este valor o termo *steps*, que também será usado no texto daqui por diante;
- L3 cache write miss;
- L3 cache read miss;
- L4 cache write miss e
- L4 cache read miss.

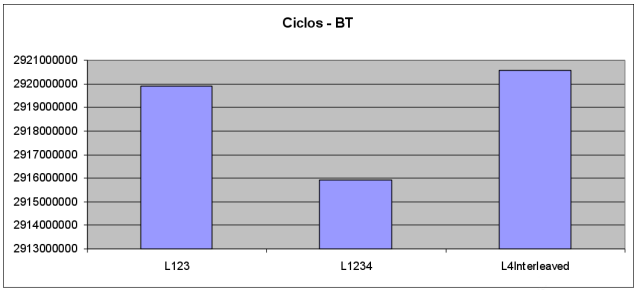

Figura 5. Números de ciclos necessários para executar o BT

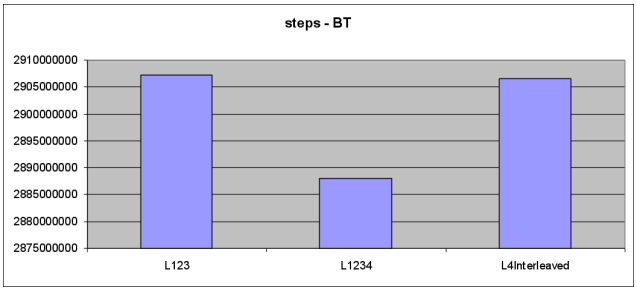

Figura 6. Números de steps necessários para executar o BT

Antes de se analisar os resultados, deve-se considerar que o desvio-padrão do número de ciclos não ultrapassou, em nenhuma das baterias de testes, o valor de

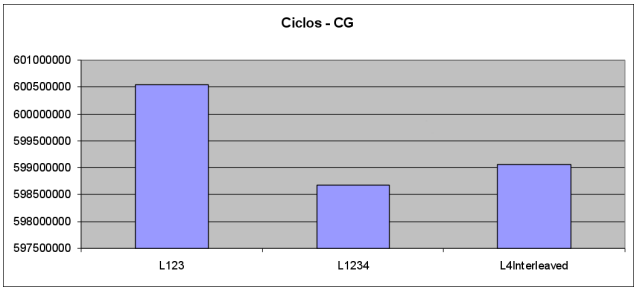

Figura 7. Números de ciclos necessários para executar o CG

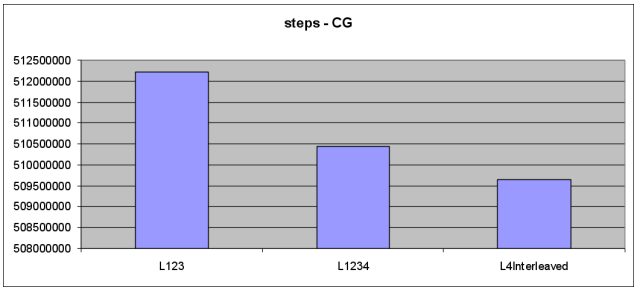

Figura 8. Números de steps necessários para executar o CG

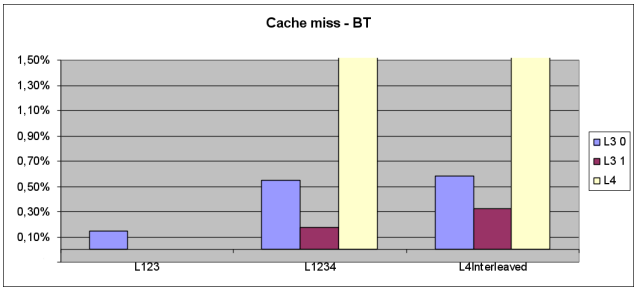

Figura 9. Taxa de cache miss ao executar o BT

1000 (0,0001%); da mesma forma que o desvio-padrão do número de steps não ultrapassou o valor de 1000000, (0,1%). Levando isso em conta, são consideradas relevantes as diferenças de resultados encontradas nos gráficos apresentados.

Nos gráficos da Figura 5 e da Figura 6, verifica-se que houve uma diferença de vários desvios-padrões no número de ciclos e de steps ao se executar o BT nas diferentes arquiteturas. A arquitetura L1234 teve o menor tempo de execução, provavelmente devido à menor contenção pelo acesso a cada cache L3, além do próprio tempo de acesso ser menor. Mesmo cada módulo de cache L3 tendo

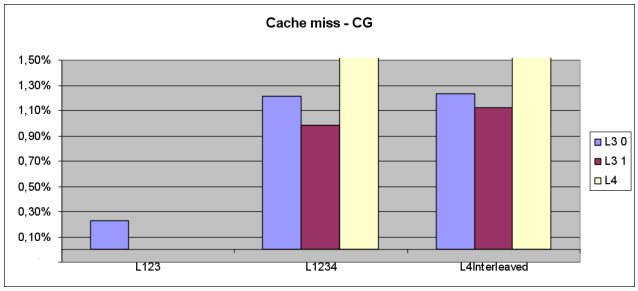

Figura 10. Taxa de cache miss ao executar o CG

um quarto da capacidade da L3 da arquitetura L1234, aparentemente a menor contenção e o acesso mais rápido foram determinantes no tempo de execução. A arquitetura L4Interleaved teve um tempo de execução próximo ao da L123, provavelmente devido ao seu maior tempo de acesso à cache L3.

Na Figura 7 e na Figura 8, verificamos um comportamento mais interessante: ambas arquiteturas com cache L4 tem desempenho melhor que aquela sem o quarto nível; porém o número de ciclos da L4Interleaved é um pouco maior que o da L1234 e número de steps é menor. A causa mais provável para isto é o entrelaçamento da hierarquia de cache. Provavelmente, o número de acessos à memória, na L4Interleaved, é menor. Contudo, como cada acesso à L3 tem um custo maior, o seu número de ciclos acabou sendo superior. Provavelmente, neste caso, fosse melhor utilizar no processador uma unidade que fosse responsável por fazer as buscas em paralelo em ambos os módulos de cache L3, reduzindo o tempo de acesso para o tempo de acessar apenas uma memória.

Por fim, na Figura 9 e na Figura 10, verificamos que a taxa de cache miss das memórias L3 aumentou – o que era esperado de uma memória quatro vezes menor. Também percebeu-se que a taxa de cache miss no programa CG é maior que no BT, o que também era esperado, considerando que o CG visa justamente a estressar as memórias cache. Por último, observou-se um valor muito alto para a taxa de cache miss da memória L4. Considerando que o intervalo entre dois acessos consecutivos é relativamente longo – já que é necessário ocorrer miss nas memórias L1, L2 e L3, para então a L4 ser acessada –, provavelmente o que acontecesse entre dois acessos consecutivos é a sobrescrita do dado que estava armazenado naquela cache, o que explicaria esse comportamento. Outra possível explicação seria o fato da cache L4 ser compartilhada por oito núcleos, escrevendo nela continuamente. O que difere essa situação de quando havia uma L3 compartilhada entre os oito núcleos é que a L4 compartilhada tem 16 MB de memória, metade da L3 compartilhada na arquitetura L123.

5. Conclusões

Este trabalho avaliou a diferença de desempenho de arquiteturas de três e quatro níveis de memória cache, além de propor uma hierarquia entrelaçada de memória. Verificou-se que, nas situações simuladas, é mais interessante adicionar um quarto nível de cache e dividir o terceiro em dois módulos do que simplesmente aumentar o tamanho da cache L3. Isso se deve ao menor tempo de acesso e à menor contenção pelo acesso à L3 presente nas arquiteturas com cache L4 simuladas. Fica como possível trabalho futuro avaliar se uma cache L4 com tamanho maior – 32 megabytes, por exemplo – teria uma taxa de cache miss tão alta.

Referências

[1] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical report, Technical Report NAS-95-020, NASA Ames Research Center, 1995.

[2] L. Dagum, R. Menon, and S. Inc. OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998.

[3] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *ISCA '98: 25 years of the international symposia on Computer architecture (selected papers)*, pages 255–262, New York, NY, USA, 1998. ACM.

[4] H. Jin, M. Frumkin, and J. Yan. The OpenMP implementation of NAS parallel benchmarks and its performance. *NASA Ames Research Center, Technical Report NAS-99-011*, 1999.

[5] Y. Li and J. Henkel. A framework for estimation and minimizing energy dissipation of embedded hw/sw systems. In *DAC '98: Proceedings of the 35th annual Design Automation Conference*, pages 188–193, New York, NY, USA, 1998. ACM.

[6] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[7] Y. Zhao, I. Raicu, and I. Foster. Scientific workflow systems for 21st century, new bottle or new wine? In *SERVICES '08: Proceedings of the 2008 IEEE Congress on Services - Part I*, pages 467–471, Washington, DC, USA, 2008. IEEE Computer Society.