Computação de Alto Desempenho em Plataforma *Windows*

Francisco Heron de Carvalho Junior

Departamento de Computação – Universidade Federal do Ceará (UFC)
Campus Universitário do Pici– Av. Mister Hull s/n – CEP 60.455/760
Fortaleza – CE – Brazil

heron@lia.ufc.br

Resumo. *Nos últimos anos, a indústria do software tem abordado com ênfase o nicho de aplicações de computação de alto desempenho. Além de apresentar o contexto geral das tecnologias que hoje viabilizam estas aplicações, este tutorial focaliza as principais características do WCCS (Windows Compute Cluster Server), a versão do sistema operacional Windows voltado a clusters, em especial no que diz respeito ao desenvolvimento de aplicações que exploram o paralelismo inerente a estas arquiteturas.*

1. Introdução

Nos últimos anos, tem emergido o interesse da indústria do *software* por aplicações de computação de alto desempenho. Tradicionalmente, estas são oriundas de diversas áreas das ciências computacionais e engenharia, possuindo impacto significativo no potencial de desenvolvimento científico e tecnológico dos que detém tecnologias capazes de viabilizá-las, notadamente instituições acadêmico-científicas e indústrias em diversas áreas. Têm contribuído para este novo contexto a proliferação da implantação de arquiteturas de computação paralela que conciliam baixo custo e alto desempenho, comparável a arquiteturas proprietárias ditas de "supercomputação", como *clusters* **[1]** e grades computacionais **[8]**. A maior facilidade em dispôr de arquiteturas de alto desempenho, além de contribuir para elevação direta do aumento do número de usuários de computação de alto desempenho, tem favorecido o surgimento de novas aplicações, as quais alimentam ainda mais o crescimento da base de usuários interessados neste nicho de aplicações, formando um mercado emergente e interessante sob a perspectiva da indústria do *software*. Deve-se ressaltar que, dentre as aplicações que têm emergido, muitas são de interesse comercial estratégico, como muitas das que apresentaremos na Seção 2.

Embora a evolução tecnológica das arquiteturas de computação de alto desempenho sob a perspectiva do *hardware* tenha obedecido a Lei de Moore[1], o mesmo não tem ocorrido com a evolução das tecnologias de *software* capaz de explorar o desempenho máximo destas arquiteturas **[7]**. Este fato tem sido observado e discutido pela comunidade científica desde o final da década de 1980, motivando inclusive a implantação, em 1989, do CRPC (*Center for Research on Parallel Computation*)[2], envolvendo várias instituições de pesquisa a nível internacional, financiado pela NSF (*National Science Foundation*), instituição de fomento dos EUA. Este fato é considerado por muitos pesquisadores como um marco **[6]**, tendo em vista o início da orquestração de esforços de importantes grupos acadêmicos e industriais para pesquisa e desenvolvimento de novas tecnologias de *software* para computação de alto desempenho, notadamente visando as áreas de algoritmos paralelos e programação paralela, tornando estas tecnologias mais próximas dos cientistas. Assim, durante a década de 1990, surgiram ferramentas

[1] Ainda em 1965, Gordon Moore, fundador da Intel, em 1965, previu que o desempenho dos processadores duplicaria a cada dezoito meses.
[2] http://www.crpc.rice.edu/.

portáveis para programação em arquiteturas de memória distribuída, como o PVM (Parallel Virtual Machine) **[9]** e MPI (Message Passing Interface) **[11]**, e memória compartilhada, como OpenMP **[12]**, além da importante evolução de bibliotecas científicas de propósito específico **[6]**, como ScaLAPACK, PETSc, NWChem, dentre outras. Até então, cada arquitetura fornecia seu conjunto próprio de ferramentas proprietárias, tornando ainda mais custoso sua implantação e manutenção. A iniciativa *High-Performance Computing and Communications* (HPCC)[3], incluindo a indústria, também teve importantes contribuições neste contexto. Vale ressaltar que o CRPC cessou suas operações oficialmente no ano 2000, sendo hoje sucedido pela iniciativa Hi-PerSoft (*Center for High Performance Software Research*), liderado pela Universidade de Rice.

Apesar do progresso no sentido da padronização do *software* sobre arquiteturas de computação de alto desempenho, de fato mais um fator estimulante de sua proliferação devido a maior usabilidade alcançada por estas arquiteturas, pouco progresso prático foi realizado para a construção de artefatos de programação paralela capazes de reconciliar os requisitos de eficiência e portabilidade com os requisitos de generalidade e alto nível de abstração e modularidade necessários para integrá-los aos artefatos de desenvolvimento de *software* comumente adotados no desenvolvimento de aplicações comerciais **[3]**. No contexto de aplicações de computação de alto desempenho, tendo em vista a voraz demanda pelo aproveitamento máximo da capacidade de processamento de arquiteturas paralelas, a introdução de mecanismos de abstração e modularidade comumente usados em artefatos de programação convencionais possui sobrecarga negativa sobre o desempenho que não pode ser desprezada. Contribui para isso a grande heterogeneidade das arquiteturas de alto desempenho, onde o desempenho de programas que exploram características peculiares destas é bastante sensível a mudanças. Com o surgimento de aplicações sobre grades computacionais e a consequente necessidade de maior integração entre aplicações, notadamente em ambiente distribuído, a necessidade por ferramentas de programação de larga escala capazes de explorar o pico desempenho de arquiteturas paralelas, tem se tornado um requisito crescentemente importante **[5]**.

Reconhecendo então a amplitude e importância do nicho de aplicações abordadas pela computação de alto desempenho, e aproveitando-se da necessidade iminente da evolução das tecnologias de *software* voltadas a estas arquiteturas, a Microsoft tem lançado nos últimos anos um programa específico para esta área, a qual culminou no lançamento do *Windows Compute Cluster Server* (WCCS), sistema operacional voltado a simples implantação, configuração e gerenciamento de *clusters*. A Microsoft tem compartilhado seu interesse com outras indústrias de *software* e comunidade acadêmica internacional para construção de tecnologias de suporte a aplicações de alto desempenho sobre arquiteturas paralelas, visando tornar tais tecnologias disseminadas na indústria, extrapolando o interesse acadêmico e científico. Este tutorial tem por objetivo apresentar as principais características da tecnologia para computação de alto desempenho aplicadas ao ambiente Windows, procurando reconhecer pontos de possíveis evoluções que podem vir a se tornar temas de pesquisa de relevância acadêmica e industrial.

Na Seção 2, apresentaremos as características gerais de aplicações de computação de alto desempenho. Na Seção 3, as tecnologias existentes para viabilizar esta classe de aplicações são introduzidas, discutindo-se as limitações destas. Na Seção 4, a arquitetura geral do *Windows Compute Cluster Server* é apresentada, incluindo os conceitos necessários para sua compreensão. Na Seção 5, através de dois exemplos, apresentaremos como programas paralelos podem ser desenvolvidos no WCCS, utilizando-se o ambiente de desenvolvimento Visual Studio 2005. Na Seção 6, são delineados os próximos passos previstos pela Microsoft na área de computação de alto desempenho. Finalmente, apresentamos nossas considerações finais na Seção 7.

[3] http://www.hpcc-usa.org/.

2. Aplicações de Computação de Alto Desempenho

A existência e evolução de qualquer tecnologia é primeiramente motivada pela existência de aplicações que desta demandam. Dessa forma, faz-se útil discutir as características gerais de aplicações de área de computação de alto desempenho, de forma a identificar seus requisitos mais importantes. Tradicionalmente, estas aplicações emergem nas áreas de ciências computacionais e engenharia, de interesse tanto de instituições acadêmicas interessadas em pesquisa científica e tecnológica como de indústrias de grande porte. Como exemplo, vale enumerar várias delas:

- **Previsão climática**, através de sofisticados modelos físicos acoplados. De fato, o uso de recursos de computação de alto desempenho por centros de previsão climática ao redor do mundo é bastante disseminado. Como ponto de partida para investigar estas aplicações, recomendamos: o *site* do NCAR (*National Center for Athmospheric Research*), nos EUA, em http://www.ncar.ucar.edu/; o *site* o Centro de Previsão de Tempo e Estudos Climáticos do Instituto de Pesquisa Espaciais (CPTEC-INPE), no Brasil, em http://www.cptec.inpe.br/; e o *site* do *UK-Japan Climate Collaboration* (UJCC), em http://www.earthsimulator.org.uk/;
- **Previsão e simulação dos efeitos de catástrofes** (erupções vulcânicas, terremotos, tsunamis, tornados, furacões, etc), com o objetivo de prevenção, o que envolve o estabelecimento de planos de evacuação, obras de infraestrutura para contenção, e regras para construção de novas edificações e reforma de edificações existentes. Exemplos de centros e projetos que desenvolvem e utilizam este tipo de tecnologia são o *Sourthern California Earthquake Center*[4], nos EUA, o *APEC Cooperation for Earthquake Simulation*[5], financiado por Austrália, China, Japão, e EUA, e o NEES (*Network for Earthquake Engineering Simulation*) Grid **[18]**;
- **Fisiologia dos seres vivos**, com a finalidade de investigar a dinâmica de sistemas orgânicos de seres vivos frente ao ataque de doenças, exposição à radiação, medicamentos, etc, por meio de simulação, além de buscar *insights* sobre o entendimento de seu funcionamento **[17]**;
- **Modelagem de reservatórios de petróleo**, com o objetivo de otimizar o processo de extração deste, reduzindo custos e aumentando a segurança do material humano envolvido. Trata-se de uma tradicional área de aplicação para computação de alto desempenho, fortemente patrocinado pelas indústrias petrolíferas. Recomendamos a busca de maiores informações no site da SPE (Society *of Petroleum Engineers*) em http://www.spe.org/. No Brasil, destacam-se os esforços do CENPES (Centro de Pesquisa Leopoldo A. Miguez de Mello), da PETROBRÁS, associado em rede com várias instituições de pesquisa nacionais;
- **Dinâmica dos fluidos**, notadamente aplicada à avaliação e otimização aerodinâmica, de interesse das indústrias automobilística e aeropespacial. Recomendamos o site da NAS (*NASA Advanced Supercomputing*), em http://www.nas.nasa.gov/, como fonte de informações;
- **Problemas de otimização combinatória**, os quais envolvem algoritmos de natureza árdua, exigindo grande capacidade de processamento para cobertura de instâncias realísticas;
- **Descoberta de novos fármacos**, através da modelagem molecular para composição de compostos químicos artificiais e simulação dos efeitos destes e compostos naturais sobre organismos simulados, através de experimentos *in silico*. Por exemplo, um interessante resultado prático foi obtido pelo projeto WISDOM (*Worldwide In Silico Docking of Malaria*)[6].

[4] http://www.scec.org/
[5] http://www.quakes.uq.edu.au/ACES/
[6] http://wisdom.eu-egee.fr/

Usando uma grade computacional, conseguiu em quatro meses (01/10/2006 a 31/01/2007) analisar mais de 140 milhões de compostos químicos, sendo encontradas três famílias de moléculas que poderiam ser efetivamente utilizadas contra o parasita da Malária. De fato, há várias empresas com soluções para este tipo de aplicação, devido ao interesse econômico das indústrias farmacêuticas em reduzir custos e tempo de descoberta de novos produtos **[14]**;

- **Genômica e bioinformática**, aplicadas ao sequenciamento de sequências de DNA e a compreensão de seu significado fisiológico. Como exemplo, o *Wellcome Trust Sanger Institute*[7], em Cambridge/GBR, gera e armazena extensas bases de dados genômicos, produzidos a partir de um conjunto de *clusters* cujo poder de processamento ultrapassa 2,5 teraflops, de interesse de pesquisa biomédica, visando aprimorar a compreensão sobre a relação entre função de genes específicos, manutenção da saúde, e ocorrência de doenças. No Brasil, a bioinformática e genômica é área interdisciplinar de interesse do Laboratório Nacional de Computação Científica (LNCC), participante de várias de redes de pesquisa patrocinadas por órgãos de fomento brasileiros, como CNPq e FAPESP;
- **Engenharia financeira, econofísica e finanças quantitativas** **[15],** envolvendo a implementação de complexos modelos matemáticos, muitos dos quais oriundos de analogia com teorias físicas e envolvendo modelos estatísticos, para solução de problemas típicos da área financeira, como, por exemplo, o movimentos de preços no mercado, comumente associado ao modelo *Black-Scholes* **[16]**, e a previsão de colapso de sistemas de seguridade;
- **Mineração de dados**, visando a análise de extensas bases de dados não estuturadas para extrair informações úteis e estratégicas em domínios específicos. Técnicas de inteligência artificial são em geral empregadas, com alta demanda computacional simbólica.

A lista cima, embora não exaustiva, ajuda-nos a identificar os requisitos comuns entre estas aplicações. Em primeiro lugar, aplicações relacionadas à simulação de fenômenos físicos recorrem a sofisticados modelos matemáticos baseados em equações diferencias parciais. Para solução destas, em geral aplicam-se métodos numéricos baseados na discretização do domínio espacial, como *diferenças finitas*, *elementos finitos* e *volumes finitos*. Estas resultam em sistemas de equações lineares para cuja solução podem ser aplicadas várias técnicas, cuja adequação varia de acordo com os padrões de ocorrência de zeros na matriz característica do sistema. Estas técnicas requerem grande velocidade de processamento para cálculos em ponto-flutuante, o que envolve também o suporte adequando no nível de linguagens de programação. Não é a toa que um dos principais parâmetros que caracterizam arquiteturas de processamento de alto desempenho é a *quantidade de operações de ponto flutuante realizadas por segundo*, ou FLOPs.

Uma demanda crescente que merece ser destacada diz respeito à utilização de grandes bases de dados em simulações, provenientes de medições reais e/ou outras simulações acopladas. Tais bases de dados têm crescido exponencialmente nos últimos anos, em taxas superiores à Lei de Moore, a qual tem governado a evolução da capacidade do *hardware*. O *Wellcome Trust Sanger Institute*, citado como exemplo de aplicação em genômica e bioinformática, é um caso típico. Conjectura-se que esta base de dados continuará seu crescimento exponencial durante muitos anos. Este fato, observado nesta e muitas outras aplicações oriundas, sobretudo, da física de alta energia, astronomia, previsão de clima, e seismologia, dificulta substancialmente a capacidade de análise de dados por meios automáticos, tendo em vista que a maioria dos algoritmos mais eficientes envolvidos são polinomiais de ordem maior que dois, além de tornar inviável que o cientista ou engenheiro copie e mantenha bases de dados em suas máquinas locais, exigindo ferramentas que ofereçam novos meios destes interagirem diretamente com os dados em seus locais de armazenamento. Ainda, tendo em vista que grandes bases de dados científicos são

[7] http://www.sanger.ac.uk/

armazenadas em meios não-voláteis, outro fator preocupante é o crescimento da capacidade de armazenamento em meios físicos não-voláteis nos últimos dez anos, uma ordem de magnitude mais rápido que a largura de banda de entrada e saída destes mecanismos (100X contra 10X). De fato, em uma implementação ingênua, uma aplicação que faz uso de uma extensa base de dados pode ter quase todo o tempo de execução dominado por operações de entrada e saída.

Além das questões relativas ao armazenamento e recuperação eficientes de dados científicos, a semântica destes dados também tem emergido como um fator relevante a ser considerado. Para isso, são ainda necessárias tecnologias que permitam a visualização e interpretação de grandes volumes de dados por cientistas, de forma que estes possam extrair as informações efetivamente necessárias para uma nova simulação ou mesmo para geração e verificação de hipóteses científicas. Técnicas de mineração de dados, notadamente associadas a técnicas de inteligência computacional, são comumente usadas, exigindo grande poder computacional.

A necessidade de integração de vários modelos acoplados é outro requisito que tem se tornado crescentemente importante em aplicações de computação de alto desempenho, enfatizando seu caráter heterogêneo. Aplicações de simulação multifísica são casos típicos. Um exemplo tradicional são sistemas acoplados de simulação climática, onde vários modelos de sistemas envolvidos, tais como atmosfera, oceano, terra, e gelo, são orquestrados por meio de um modelo acoplador, responsável por computar as contribuições de cada modelo no modelo global de clima. De fato, os diferentes modelos envolvidos podem estar codificados em linguagens de programação diferentes, recaindo-se sobre as naturais dificuldades de mapeamentos entre representações de estruturas de dados entre linguagens. Além disso, no caso de modelos implementados com técnicas que pressupõem a discretização de domínio, diferentes modelos podem assumir diferentes formatos de elementos e/ou resolução de malha, sendo necessário o mapeamento do espaço entre um domínio e outro, o qual pode se tornar uma operação bastante custosa em implementações ingênuas. Com a introdução de computação distribuída na *internet*, desde o advento da tecnologia *Web* até a viabilização de grades computacionais, o compartilhamento *online* de modelos entre instituições e laboratórios de pesquisa tem sido explorado, especialmente através de portais científicos ou mesmo de serviços de computação espalhados na rede usando grades computacionais ou tecnologias de acesso a serviços distribuídos, como *web services*.

Um conceito comum em aplicações de alto desempenho é a “corretude por reputação”, estágio alcançado por um *software* científico o qual tem sido executado com sucesso durante um longo tempo, em múltiplos contextos arquiteturais. Há vários exemplos de código legado, em geral encapsulado em bibliotecas científicas nos mais diversos domínios, usados há mais trinta anos com sucesso em aplicações de computação de alto desempenho. Portanto, é natural para cientistas e engenheiros a divulgação de *software* primeiramente para validação e testes, visando à futura promoção deste a *software* finalizado, presumivelmente confiável e robusto, pela própria comunidade científica. Assim, o controle de versão de código faz-se útil no meio científico, favorecendo a evolução do *software* ao modo usado pelos cientistas durante décadas.

3. Tecnologias de Suporte a Computação de Alto Desempenho

Aplicações de alto desempenho possuem requisitos particulares que demandam por tecnologias específicas, as quais provêem de desenvolvimentos em três áreas distintas: algoritmos abstratos, arquiteturas de computadores, e ambientes de *software*. A primeira diz respeito às investigações de algoritmos eficientes, do ponto de vista de complexidade abstrata, para solução dos diversos problemas que ocorrem em aplicações em ciências computacionais e engenharia, e que demandam por implementação computacional de alto desempenho. Por exemplo, algoritmos iterativos para solução de sistemas de equações diferenciais parciais têm sido desenvolvidos e aprimorados ao longo dos anos, como *Gauss-Seidel*, *Red-Black*, *Successive Over Relaxation*,

métodos *multigrid,* fatoração LU, etc. Algoritmos para geração de malhas, também necessários à implementação do método de elementos finitos, são outros exemplos importantes. Frequentemente, algoritmos para aplicações de computação de alto desempenho abordam a questão da paralelização como prioritária. De fato, em muitos casos, algoritmos "mais paralelizáveis" são preferidos em relação a algoritmos de menor complexidade, porém "menos paralelizáveis" devido a sua melhor escalabilidade sobre arquiteturas paralelas. As duas demais áreas serão especialmente destacadas nos parágrafos seguintes.

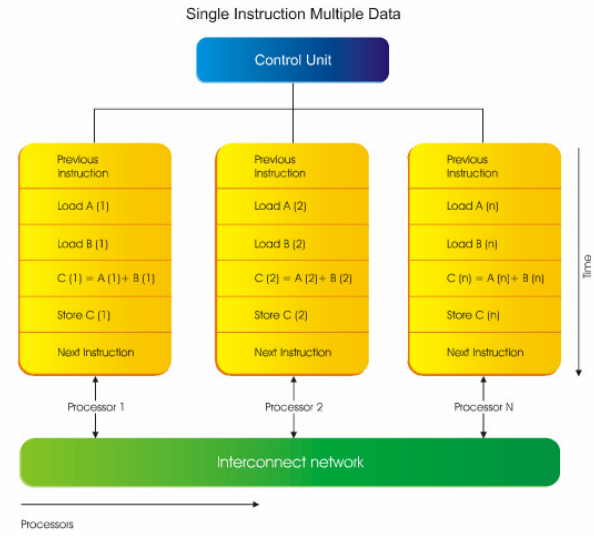


Figura 2. Arquitetura SIMD

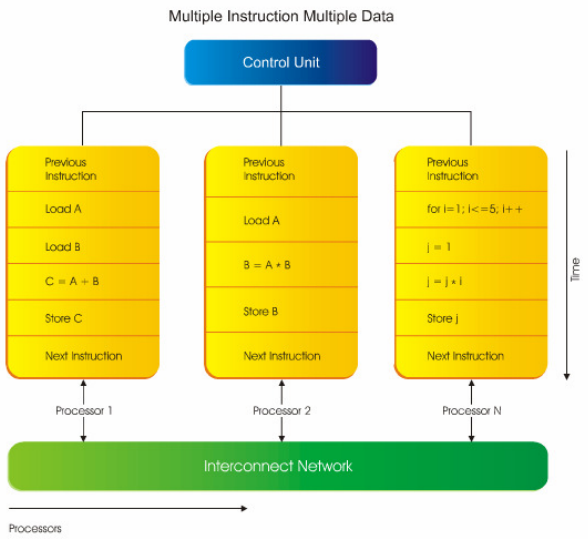


Figura 2. Arquitetura MIMD

Arquiteturas de Computadores de Alto Desempenho

Sobre este assunto, há uma longa história a ser contada, a qual remonta ao ano de 1972 quando Seymour Cray fundou a *Cray Research*, responsável pela implantação do primeiro dito supercomputador no Los Alamos National Laboratory (LANL), EUA, em 1976, ao custo de 8,8 milhões de dólares. Esta máquina era capaz de executar 160 megaflops (milhões de operações de ponto flutuante por segundo), com 8MB de memória principal. Naquela época, tal desempenho superava em ordens de magnitude o desempenho da mais rápida das máquinas existentes do mercado. Os computadores Cray dominaram o cenário da supercomputação durante vários anos, rivalizando com supercomputadores japoneses de arquitetura vetorial, em especial produzidos pela NEC, iniciando uma marcante competição entre indústrias japonesas e norte-americanas.

Arquiteturas vetoriais dominaram a supercomputação nas décadas de 1970 e 1980. Ditas de arquitetura SIMD *(Single Instruction Multiple Data*), são capazes de executar uma operação simultaneamente sobre cada elemento de um conjunto de dados (Figura 2). Tais arquiteturas demandam por linguagens específicas e compiladores "paralelizantes". Durante a década de 1980, a investigação sobre paralelização automática de programas, em especial a vetorização de *loops*, foi intensa. Entretanto, arquiteturas vetoriais mostram-se pobremente escaláveis, apesar de seu alto custo de implantação, operação e manutenção. A ampliação da capacidade de processamento pela adição de unidades de processamentos era uma possibilidade inviável.

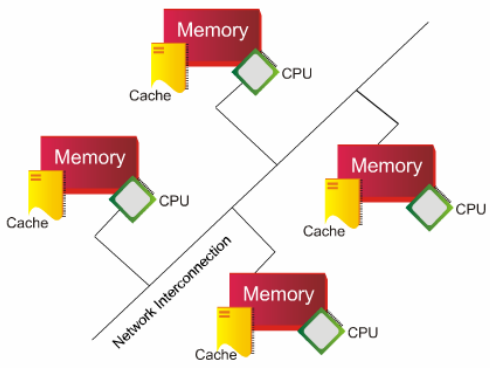


Figura 3. Massive Parallel Processors

Alternativas a computação vetorial emergiram a partir de meados da década de 80, em especial as arquiteturas de memória distribuída. Estas são ditas de

arquitetura MIMD, onde a cada processador corresponde uma linha de instrução independente que opera sobre um conjunto de dados disjunto (Figura 2). Entretanto, tais arquiteturas somente tornaram-se viáveis com o surgimento de interfaces de comunicação de alto desempenho, em especial proprietárias, levando ao surgimento de arquiteturas ditas massivamente paralelas, as chamadas MPP (*Massive Parallel Processors*). Exemplos clássicos destas arquiteturas são: Intel Paragon, Cray T3D e T3E, Thinking Machine CM5, IBM SP2, ASCI Red e Sun HPC. Embora possuindo maior escalabilidade, além de suplantar máquinas vetoriais em desempenho bruto, MPPs exigiam modelos explícitos de programação paralela, em geral baseados em troca de mensagens, a fim de aproveitar ao máximo o seu desempenho.

Durante este período, surgiram ainda as arquiteturas vetoriais de multiprocessamento, admitindo o compartilhamento de memória entre um conjunto de processadores de tecnologia vetorial, como a família Cray X-MP. Estas arquiteturas, ilustradas na Figura 4, também são de arquitetura MIMD, porém as várias linhas de instrução podem operar sobre conjuntos possivelmente não disjuntos de dados. A arquitetura SMP (*Shared Memory Processors*) foi utilizada nos supercompadores Cray e outros fabricantes durante a década de 80 e início dos anos 90. Motivaram ainda arquiteturas SMP com processadores escalares, as quais foram bastante utilizadas em laboratórios de pesquisa, devido ao custo menor. Entretanto, a escalabilidade de SMP's é limitada pela capacidade do barramento de suportar a contenção de memória causada por um grande número de processadores, o qual, por este motivo, não poderia ultrapassar as dezenas.

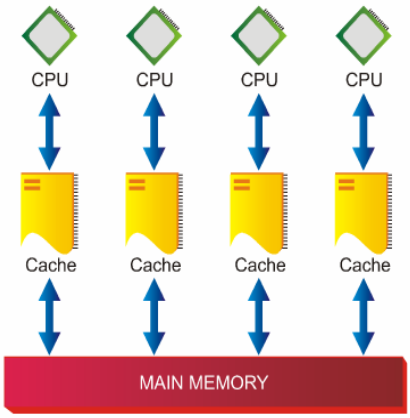


Figura 4. Multiprocessadores (SMPs)

No início da década de 90, iniciaram-se as iniciativas de explorar o paralelismo em redes de estações de trabalho (NOWs, ou *Networks of Workstations*) **[22]**, ilustradas na Figura 5. De fato, por questões de barateamento de custos na fabricação de MPPs, cuja escala de mercado é pequena, os processadores usados em estações de trabalhos eram de tecnologia semelhante à aplicada em MPPs, o que resultou em um contexto indesejável. Devido ao tempo necessário para que indústrias fabricantes de MPPs desenvolvessem redes de comunicação de alto desempenho adaptadas às características peculiares de uma nova classe de processadores, a MPP baseada em processadores desta nova classe demorava certo tempo para ser lançada de fato no mercado, o qual ficou conhecido como *lag time*. Na época do lançamento, devido ao grande dinamismo deste mercado, uma nova linha de processadores já estava sendo lançada e já equipava as estações de trabalho, cujo mercado possuía maior escala e rotatividade de atualização de equipamentos, causando sempre uma indesejável defasagem na tecnologia de processadores usados nas MPPs em relação às estações de trabalho, a qual compensava a perda de eficiência devido à adoção de redes de comunicação convencionais em NOWs, as permitindo abordar nichos de aplicações antes exclusivos de MPPs.

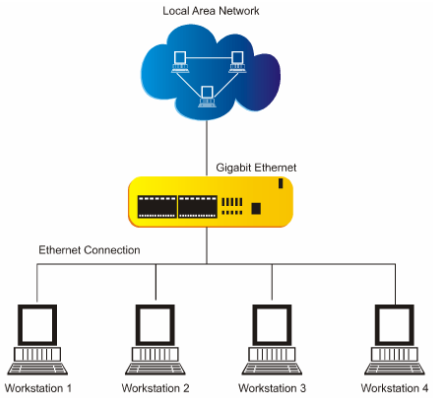


Figura 5. Network of Workstations

A partir da primeira metade da década de 90, o crescimento vertiginoso da escala do mercado de processadores para computadores pessoais (PCs) levou aos altos investimentos em pesquisa e desenvolvimento da maior fabricante naquela época, a Intel. Tais investimentos, viabilizados pela maior escala do mercado de PCs em relação ao mercado de estações de trabalho e MPPs, levaram os processadores pessoais a um desempenho que alcançou o desempenho de estações de trabalho, porém a um custo muito inferior, levando a quase extinção do mercado de es-

tações de trabalho. Então, logo surgiram os primeiros projetos que exploraram o uso de PCs conectados em redes convencionais de comunicação, do padrão *Ethernet*, para aplicação em computação de desempenho. Destaca-se como marco o projeto *Beowulf* **[21]**, conduzido por Donald Becker, em 1994, na NASA, o qual possuindo escassos recursos para aquisição de máquinas paralelas às quais demandava, resolveu construí-las a partir de *hardware* de prateleira, reusando equipamentos fora de uso. Estas arquiteturas ficaram conhecidas como *clusters* e significaram a proliferação das tecnologias de computação de alto desempenho em vários laboratórios de pesquisa espalhados pelo mundo, para os quais a tecnologia de supercomputação era muito custosa. É importante ressaltar que os *clusters* são fenômenos fundamentalmente influenciados pelo *software* que foi desenvolvido para o seu suporte. De fato, Becker e sua equipe desenvolveram interfaces de comunicação capazes de explorar ao máximo o desempenho de sua arquitetura. Para isso, sendo de fundamental importância o uso de código livre disponibilizado pelas emergentes distribuições do sistema operacional *Linux*.

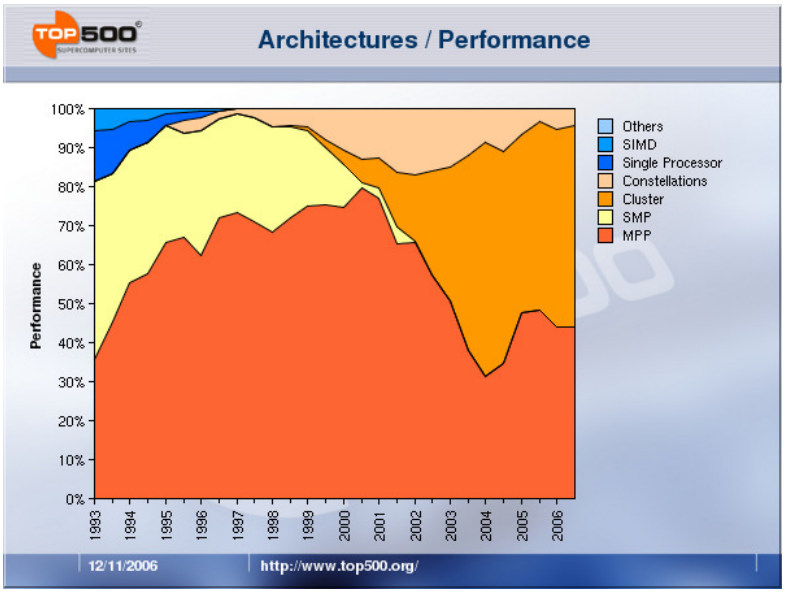


Figura 6. Participação das principais arquiteturas de supercomputação na lista Top 500 (extraído de http://www.top500.org)

Segundo o site top500 (http://www.top500.org), onde são enumeradas as quinhentas arquiteturas de supercomputação de maior desempenho segundo *benchmarks* padrões, de 1998, quando *clusters* começaram a aparecer nesta lista, até meados do ano de 2003, o uso de *clusters* cresceu exponencialmente, chegando a dominar, neste mesmo ano, 60% das posições nesta lista. Nos últimos anos, a participação de *clusters* tem estabilizado, mantendo-se no patamar de 55%, em parte devido ao crescimento da participação de arquiteturas NUMA (*Non-Uniform Memory Access*), as quais trouxeram maior escalabilidade às arquiteturas de memória compartilhada.

No final da década de 1990, a expansão e incremento de desempenho de infraestruturas de **comunicação** de alcance global, a *internet*, motivou a idéia da implementação, no topo destas, de uma infraestrutura de **computação**, capaz de virtualizar o conjunto de recursos computacionais disponíveis na internet com a finalidade torná-las disponíveis a aplicações que demandam por alta carga computacional. A esta tecnologia deu-se o nome de *grades computacionais*, devido à analogia com a infraestrutura de distribuição de energia elétrica, a qual apresenta os recursos a seus usuários de maneira transparente à tecnologia nesta implementada. Desde então, a pesquisa interdisciplinar para viabilização de grades disseminou-se, motivando o surgimento de vários projetos de arquiteturas de computação baseadas nesta tecnologia emergente. Evolutivamente, espera-se mover-se das grades de propósito específico para as grades de propósito geral, onde diversas tecnologias possam coexistir de maneira transparente ao usuário. Um caminho longo a seguir, portanto. Com relação ao suporte a execução paralela, a alta latência da *internet* ainda não torna possível a sincronização no topo desta de processos fortemente acoplados, característicos de *clusters* e, especialmente, MPPs. Em geral, o paralelismo em grades é explorado por meio de padrões de paralelismo onde a necessidade de comunicação entre tarefas é mínima ou mesmo inexistente, como é o caso do padrão *bag-of-tasks*. Entretanto, espera-se que esta barreira tecno-

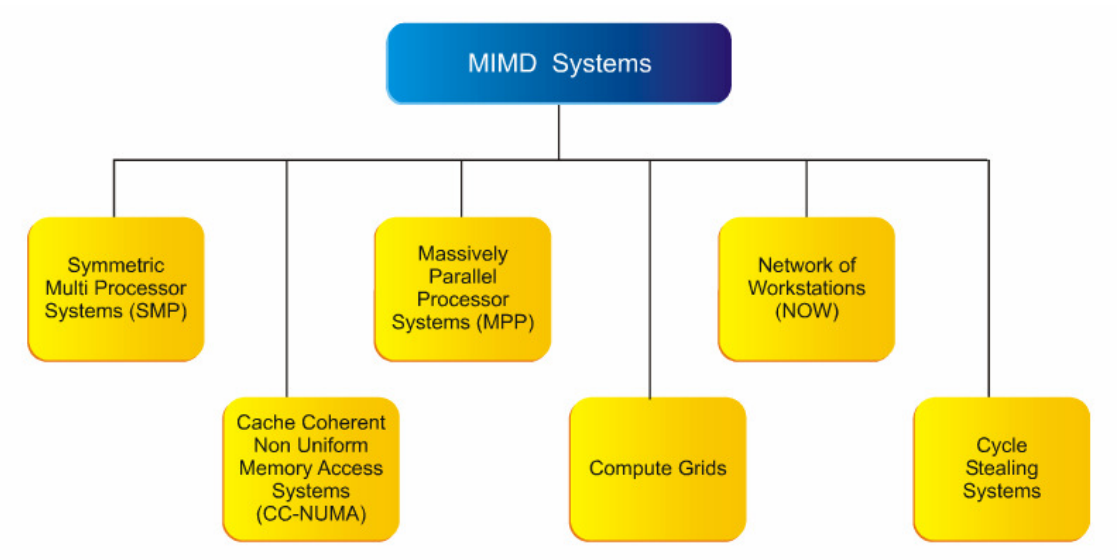


Figura 7. Classes Contemporâneas de Arquiteturas MIMD

lógica seja vencida ao longo das próximas décadas, com a evolução da infraestrutura de comunicação sobre as quais se assentam as tecnologias de grades.

A Figura 7 resume as arquiteturas de computação de alto desempenho contemporâneas. Incluem-se somente as arquiteturas MIMD, mas isso não significa que a tecnologia vetorial perdeu o significado tecnológico. De fato, processadores modernos incluem extensões vetoriais, como SSE e SSE2 (*Streamming SIMD Extensions*) da Intel, e 3DNow! da AMD. Estes possuem ainda extensões superescalares, as quais permitem a execução simultânea de instruções de forma transparente aos programadores, além de suporte ao paralelismo em nível de *threads* (*Hyper Threading* e *Core Duo* da Intel, e X2 da AMD). De fato, arquiteturas modernas de computação de alto desempenho exploram múltiplas hierarquias de processamento paralelo e memória, tornando uma tarefa cada vez mais complexa a exploração eficiente de seu potencial de desempenho.

Software para Arquiteturas de Computadores de Alto Desempenho

A heterogeneidade das arquiteturas de computação de alto desempenho constitui uma dificuldade inerente ao desenvolvimento de *software* capaz de explorar o seu desempenho. Para estas arquiteturas, o *hardware* tem evoluído de maneira independente ao *software*, e em um ritmo muito mais acelerado, especialmente devido ao maior investimento da indústria do *hardware*. Este é um problema central que tem sido abordado desde o início da década de 80. Em 1989, motivou o surgimento do CRPC (*Center for Research on Parallel Computation*), iniciativa envolvendo várias instituições de pesquisa a nível internacional, financiado pela NSF (*National Science Foundation*), dos EUA. Este fato é considerado por muitos pesquisadores como um marco **[6]**, tendo em vista o início da orquestração de esforços de importantes grupos acadêmicos e industriais para pesquisa e desenvolvimento de novas tecnologias de *software* para computação de alto desempenho. Embora resultados importantes tenham sido alcançados, estes foram aquém da expectativa inicial dos pesquisadores envolvidos.

O *software* de suporte ao paralelismo tem merecido atenção ao longo dos anos. Vários modelos de programação paralela têm sido propostos **[20][6]**, porém poucos destes têm tido uso disseminado. Bibliotecas pré-paralelizadas de propósito especial, como LAPACK, ScaLAPACK, PETSc, NWChem, dentre outras, e interfaces de passagem de mensagens, como MPI e PVM, tem sido as mais utilizadas, por oferecem o melhor aproveitamento do potencial de arquiteturas de *clusters* e MPPs. De fato, mecanismos mais abstratos pecam por não serem suficientemente flexíveis para múltiplos contextos arquiteturais, os quais muitas vezes envolvem peculiaridades inerentes a uma arquitetura específica. Tal fato torna o desempenho de tais mecanismos inferior ao

desempenho de interfaces de mais baixo nível. Analisando as ferramentas existentes de suporte ao paralelismo com relação à generalidade, portabilidade, abstração e desempenho, vemos que a conciliação entre estes não é atingida pela tecnologia vigente **[3]**. Bibliotecas de propósito específico, embora eficientes quando possuindo implementações específicas para arquiteturas particulares ou classes destas, não oferece a generalidade das interfaces de passagem de mensagens, além de muitas delas não serem tão flexíveis no suporte a configuração do paralelismo pelo programador, levando a limitações de portabilidade de desempenho. Por outro lado, interfaces de passagem de mensagens pecam pelo pouco nível de abstração, obrigando o espalhamento de interesses inerentes ao paralelismo nos módulos de programas resultantes da decomposição típica da engenharia de *software*. Entretanto, têm sido as mais abordadas devido a sua flexibilidade, desempenho, e facilidade a integração a linguagens de programação comuns como, C e Fortran, característica compartilhada com bibliotecas científicas. De fato, a conciliação de artefatos da engenharia de *software* com ferramentas de programação paralelas voltadas a computação de alto desempenho ainda é um desafio aos pesquisadores desta área.

A crescente necessidade de integração de múltiplos modelos, possivelmente desenvolvidos por equipes distintas em laboratórios ou conjuntos de laboratórios em cooperação, torna também o requisito de integração e reuso de *software* importante para aplicações de alto desempenho, como motivado na Seção 2. A concepção desta forma de cooperação característica do meio científico tem sido certamente impulsionada pelas possibilidades advindas das novas tecnologias de serviços *Web* e das grades computacionais. De fato, há portais científicos através dos quais pesquisadores e usuários podem fazer uso de modelos implementados e disponibilizados como serviços na *Web*, muitos oferecendo inclusive os próprios recursos computacionais necessários à execução, e virtualizados por meio de tecnologias de grades computacionais. Um exemplo é o *NetSolve/GridSolve*[8] **[23]**. Tendo em vista estes novos requisitos, a tecnologia de componentes tem sido enxergada como uma alternativa viável de ser aplicada em aplicações de computação de alto desempenho. Destacamos então o esforço da iniciativa CCA (*Common Component Architecture*), envolvendo diversas instituições de pesquisa e acadêmicas dos EUA, patrocinadas pelo DOE (*Departament of Energy*) e NSF (*National Science Foundation*), para o desenvolvimento de tecnologia de componentes para aplicações em ciências computacionais. A primeira especificação da arquitetura CCA, herdando muitos conceitos de modelos de componentes comerciais como CORBA e COM, surgiu no início da década de 2000, motivando o desenvolvimento de vários *frameworks* compatíveis, abordando diferentes contextos de aplicação e classes de arquiteturas, como CCAffeine, XCAT, MOCCA, SciRun2, dentre outros. Outra iniciativa, desta vez européia, diz respeito ao modelo de componentes Fractal, o qual tem motivado a implementação de *frameworks* computacionais para programação em grades computacionais, como o ProActive. No Brasil, os projetos relacionados ao modelo *hash* (#) de componentes **[4]** e ao Forró, um *framework* CCA, merecem ser lembrados.

Com relação ao gerenciamento de extensas bases de dados científicas, outro requisito importante identificado na Seção 2, a comunidade científica tem proposto ao longo dos anos o uso de formatos especiais de arquivos para compartilhamento de dados científicos. Exemplos clássicos são o HDF (*Hierarchical Data Format*)[9] , o CDF (*Common Data Format*)[10], e o NetCDF (*Network Common Data Form*)[11], bibliotecas de *software* e formatos de dados independentes de máquina que suportam a criação, acesso e compartilhamento orientado a tipos de dados comuns em aplicações científicas, como matrizes e vetores. Em tempos recentes, com o surgimento e consolidação do padrão XML, extensões a este formato passaram a ser considerados para arma-

[8] http://icl.cs.utk.edu/netsolve/.
[9] http://www.srl.caltech.edu/ACE/ASC/hdf.html/
[10] http://cdf.gsfc.nasa.gov/
[11] http://www.unidata.ucar.edu/software/netcdf/

zenamento de dados científicos, notadamente tendo em vista a compatibilidade com a emergente tecnologia de serviços *web* e grades **[19]**. Representações XML para o CDF e NetCDF têm sido propostas, como o NcML[12] no caso do último. Uma representativa coleção de formatos científicos de dados é enumerada no site http://www.dpmi.tu-graz.ac.at/schloegl/matlab/eeg/. Além disso, bibliotecas de entrada e saída de alto desempenho têm sido propostas, como o MPI-IO, uma extensão a biblioteca de passagem de mensagens MPI, capaz de coordenar operações de entrada e saída paralelas em arquivos. Tendo ainda em vista reduzir o custo de comunicação de dados, especialmente em grades computacionais, técnicas que permitem a movimentação de código ao encontro dos dados armazenados em repositórios têm sido investigadas, em alternativa ao envio da base de dados através de uma rede de comunicação para processamento local. Técnicas oriundas de sistemas de bancos de dados também têm sido abordadas, como distribuição de dados, processamento paralelo de consultas, acesso a dados orientado a conjuntos, indexação inteligente, etc. Porém, tais técnicas precisam ser extendidas para suporte nativo a tipos de dados típicos de aplicações científicas, como matrizes, vetores e números complexos.

O interesse da indústria de *software* pela área de computação de alto desempenho é um fato recente. O exemplo da Microsoft, ao lançar uma versão do sistema operacional Windows Server 2003® especialmente para *clusters*, além de várias outras iniciativas, é uma evidência deste contexto. Espera-se que isso contribua para alavancar o desenvolvimento do *software* para aplicações de computação de alto desempenho. Recomendamos o site http://www.winhpc.org/ como fonte de informações sobre os passos da indústria de *software* em HPC.

4. O *Windows Compute Cluster Server*

Na Seção 2, foram introduzidas as principais características das arquiteturas de computação utilizadas, no passado e no presente, em *computação de alto desempenho*. Dentre estas, *clusters* motivaram uma disseminação jamais vista das tecnologias de computação paralela para um mercado eminentemente composto por laboratórios científicos e de engenharia em instituições acadêmicas e indústrias interessadas em aplicações que exigiam computação massiva. Com isso, a participação dos *clusters* na lista do Top500 cresceu exponencialmente entre os anos de 1998 e 2003, quando então atingiu o pico de 68% das entradas da lista.

Em um mercado tradicionalmente dominado por várias distribuições do sistema operacional Linux, o *Windows® Compute Cluster Server* 2003 (WCCS) tem sido proposto pela Microsoft como uma alternativa baseada na versão servidor do sistema operacional Windows® para simplificar a implantação e gerenciamento de *clusters*, de forma a reduzir os custos de quem os possuem. Estima-se que isso resulte em um crescimento ainda maior na adoção dos *clusters* nos níveis de computação pessoal e em grupo. Esta seção dedica-se a apresentar os principais conceitos da arquitetura do WCCS.

4.1. Arquitetura – Visão Geral e Conceitos Básicos

O diagrama na Figura 8 provê uma visão geral da arquitetura de *software* empregada no WCCS. Um *cluster* é formado por um **nó principal** (*head node*) e um conjunto de **nós de computação** (*compute nodes*). O nó principal pode também ser configurado como nó de computação. Por intermediação do nó principal, usuários (*User*) e administradores (*Admin*) do *cluster* podem, respectivamente, utilizá-lo e gerenciá-lo. Por *utilização*, entenda-se a atividade de disparar e monitorar **trabalhos** (*jobs*) a serem executados pelos nós de computação, enquanto o gerenciamento engloba a *configuração*, *inclusão*, e *exclusão* de nós de computação e permissões de usuários. A

[12] http://www.unidata.ucar.edu/software/netcdf/ncml/

definição de trabalho, a caracterização dos tipos de trabalho suportados, assim como os mecanismos de gerenciamento (*Job Mgmt*) e escalonamento (*Scheduling*) destes serão detalhados na Seção 4.3. A interface dos usuários e administradores com o *cluster* pode ser feito de três maneiras:

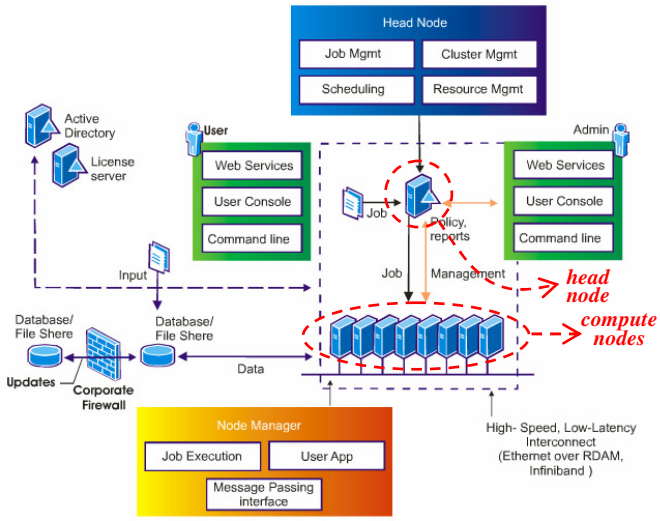


Figura 8. A Arquitetura de *Software* do WCCS

- **CCAPI.DLL**. O componente CCAPI.DLL oferece serviços para utilização e gerenciamento do *cluster* a partir de outros programas, mesmo remotos;
- **Console de Usuário** *(User Console)*. Uma interface gráfica (GUI), implementado como um *plug-in* para o *Microsoft Management Console*;
- **Linha de Comando** *(Command Line)*. *Prompt* de comando, suportando comandos básicos para gerenciamento e utilização do *cluster*: `job`, `task`, `node`, `cluscfg`, e `clusrun`.

O WCCS está integrado ao *Active Directory*®, a implementação Windows para o padrão LDAP (*Lightweight Directory Access Protocol*). O uso de seus serviços permite o gerenciamento de políticas de segurança para usuários e recursos nos nós de computação e para os trabalhos que nestes são lançados. O uso do *Active Directory* oferece a possibilidade de uma integração simples do *cluster* a uma rede corporativa pré-existente. Além disso, os nós de computação podem acessar servidores de arquivos (*Database/File Share*), onde estão localizados dados necessários ao processamento, inclusive residindo fora do domínio de segurança da rede corporativa, separada desta por *firewall* (*Corporate Firewall*).

O Microsoft Message Massing Interface (MSMPI)

Uma importante característica do WCCS é o suporte à biblioteca de passagem de mensagens MPI, a mais utilizada em aplicações de computação de alto desempenho para o desenvolvimento de programas paralelos que exploram os recursos computacionais de *clusters* e MPPs de maneira eficiente, a despeito do seu baixo nível de abstração. Com a finalidade de explorar ao máximo o desempenho de um *cluster* com tecnologia Windows, foi desenvolvido o MSMPI, uma versão do MPICH2 adaptada às características do WCCS, com a colaboração do Laboratório Nacional de Argonne (ANL), nos EUA, um renomado centro na área de tecnologia e aplicações de computação de alto desempenho. A Figura 9 apresenta a arquitetura usada na implementação do MSMPI, enfatizando o uso do protocolo *Winsock Direct*, o qual permite sobrepassar à pilha do

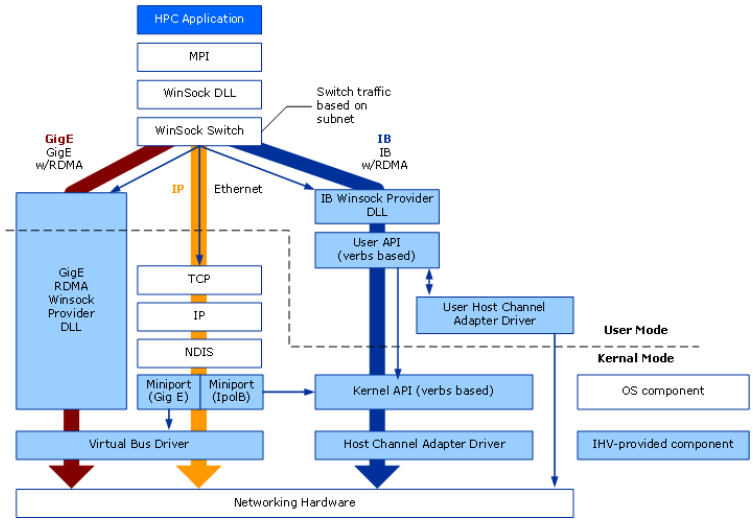


Figura 9. Winsock Direct Architecture

TCP/IP usando RDMA (*Acesso Direto a Memória Remoto*), com a finalidade de alcançar níveis elevados de desempenho. Qualquer tecnologia de interconexão que disponibilize um provedor *Winsok Direct* pode ser usada com MSMPI, como (10)-*Gigabit Ethernet*, *Infiniband* e *Myrinet*. A primeira oferece uma alternativa de melhor custo/benefício, enquanto as outras duas são ideais para aplicações muito sensíveis a latência de comunicação e que exigem grande largura de banda. De fato, os nós de computação de um *cluster* WCCS podem estar conectados através de uma interconexão de alto desempenho separada da rede corporativa. O MSMPI não introduz extensões ao padrão MPI2, sendo, portanto, compatível com outras implementações. Isso facilita a portabilidade de programas para o WCCS, além de assegurar uma tênue curva de aprendizado para o uso efetivo de um *cluster* WCCS em regime de produção.

4.2. Implantação

Os requisitos de *hardware* necessários à implantação do WCCS estão apresentados no Apêndice A. Satisfeitos estes requisitos, é essencial o planejamento da implantação, inicialmente consistindo na escolha de uma topologia de rede. O WCCS distingue entre cinco topologias de redes, envolvendo três tipos diferentes de redes que podem coexistir em certa topologia: *rede pública*, *rede privada* e *rede MPI*. A **rede pública** corresponde à rede corporativa, conectada ao nó principal e, opcionalmente, aos nós de computação. É através da rede pública que os usuários conectam-se ao *cluster* para submeter seus trabalhos, inclusive remotamente. Todo tráfego no *cluster* relacionado ao gerenciamento e implantação deve ser realizado sobre a rede pública, caso não exista uma **rede privada**. Esta é dedicada ao *cluster*, sem acesso externo, encarregando-se do tráfego relacionado ao gerenciamento, implantação e tráfego MPI, caso não esteja disponível uma **rede MPI**. Esta, portanto, responsabiliza-se por todo o tráfego MPI, em geral sobre uma interconexão de alto desempenho, como *Infiniband*, *Myrinet*, ou (10)-*Gigabit Ethernet*. Portanto, não é necessária quando os trabalhos a serem lançados no *cluster* não usarão rotinas MPI. A existência da rede MPI é essencial em aplicações de computação intensiva onde a cooperação entre tarefas exige a troca coletiva de grandes massas de dados. O tipo de rede MPI determina o tipo de aplicação suportado pelo *cluster*.

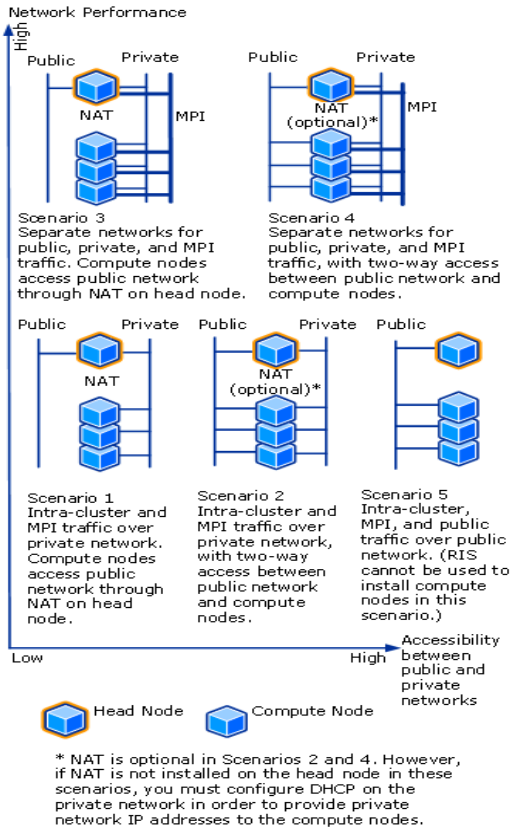


Figura 10. Topologia de Rede

As topologias apresentadas na Figura diferenciam-se pelo suporte ou não às redes privadas e MPI e pela forma de acesso a rede pública pelos nós de computação, para que estes tenham acesso aos controladores de domínio e serviços da rede pública. De fato, o acesso a serviços externos pode ser necessário em muitas aplicações. O NAT (*Network Address Translator*) é imprescindível quando os nós de computação não possuem acesso direto a rede pública (cenários 1 e 2), permitindo o acesso indireto destes, e opcional caso contrário e com existência de uma rede privada. Serviços essenciais a cada uma das redes envolvidas são o DHCP e DNS. De fato, cada interface de rede de um computador na rede pública e privada precisa de um número IP, atribuído estaticamente ou dinamicamente, e pode precisar de um mecanismo de resolução de nomes. O suporte a estes serviços (NAT, DHCP e DNS) pode ser oferecido por duas alternativas:

- O ICS (*Internet Connection Sharing*) no nó principal;

- O RRAS (*Routing and Remoting Access Services*).

Ambos oferecem serviços NAT, mini-DHCP e mini-DNS. ICS automaticamente atribui o IP estático 192.168.0.1 (SM: 255.255.255.0) ao adaptador de rede privada no nó principal (não modificável), enquanto RRAS exige atribuição manual. Dessa forma, ICS suporta até 254 nós na rede privada (rede classe C), enquanto a maior flexibilidade de RRAS permite o suporte a redes privadas de classe A e B (mais que 254 nós). Porém, RRAS não suporta o *firewall* do Windows, ao contrário do ICS. Sempre que possível, a alternativa ICS é recomendada pela Microsoft, devido a sua maior simplicidade e integração ao ambiente Windows. Porém, o ICS não implementa NAT, DHCP e DNS sobre a rede MPI. Caso seja necessário para a versão de MPI adotada, deve-se utilizar algum mecanismo compatível com esta. Recomendações especiais existem para o caso onde a rede corporativa faz uso de IPsec (protocolo IP seguro), o qual provê comunicações seguras sobre o protocolo IP por meio de técnicas de criptografia.

O processo de instalação de um *cluster* WCCS é bastante automatizado, facilitando sua implantação. É dividida nas fases de instalação do nó principal e dos nós de computação. A instalação dos últimos pode ser feita de forma *manual* ou *automática*. Esta última prescinde da existência de uma rede privada, além da instalação e configuração do RIS (*Remote Installation Services*) durante a instalação do nó principal. Deve-se ainda incorporar o nó principal a um domínio existente do *Active Directory*. Cada nó de computação adicionado deve pertencer ao mesmo domínio e deve ser explicitamente aprovado pelo administrador, o qual é ainda responsável por configurar as permissões de usuários e administradores do *cluster*. Durante a instalação do nó principal, o instalador deve ainda preocupar-se com a configuração do *Firewall* do *cluster*, o qual pode ser opcionalmente habilitado na interface pública de cada um dos nós, caso o nó principal não seja um controlador de domínio. O *Firewall* é automaticamente desabilitado nas interfaces privadas e MPI dos nós. Os detalhes do processo de instalação de nós principal e de computação não serão fornecidos neste documento, mas estão publicamente disponibilizados[13].

4.3. Utilização do Cluster – Gerenciamento de Trabalhos e Tarefas

Uma vez implantado, usuários com permissão podem lançar trabalhos no *cluster* WCCS. Um **trabalho** (*job*) é uma requisição de recursos submetida ao **escalonador de trabalhos** (*Job Scheduler*) que contém ou conterá uma ou mais **tarefas** (*task*). Uma tarefa não pode existir sem estar associada a um trabalho. Um trabalho pode conter uma única tarefa. O escalonador de trabalho é o serviço responsável por colocar trabalhos e tarefas em filas, alocar recursos, despachar tarefas para nós de computação, e monitorar o estado de trabalhos, tarefas, e nós. A ordem dos trabalhos na fila obedece a uma **política de escalonamento**, as quais serão descritas adiante. Para um determinado trabalho, a ordem das suas tarefas na fila segue a política FCFS (*First-Come*, *First-Served*), a menos que dependências entre as tarefas ditem o contrário.

A interface do usuário com escalonador de trabalhos pode ser feita através de uma GUI Win32, chamada *Compute Cluster Job Manager*, através do componente COM CCAPI.DLL, ou através do CLI (*Command-Line Interface*), um *prompt* de comandos. Neste último, os comandos relacionados ao gerenciamento de trabalhos e tarefas são `job` e `task`, dentre os cinco suportados. Os demais se referem às atividades de gerenciamento e monitoração do *cluster*. De fato, os comandos `job` e `task` são múltiplos comandos, suportando vários operadores. O significado destes operadores, descritos junto com o ciclo de vida de trabalhos e tarefas, provêem uma visão didática das operações permitidas ao usuário do *cluster* sobre trabalhos e tarefas, também válidas para os acessos por meio da interface gráfica e do componente CCAPI.DLL.

---

[13] Microsoft TechNet (http://technet.microsoft.com). Procurar “Windows Compute Cluster Server”.

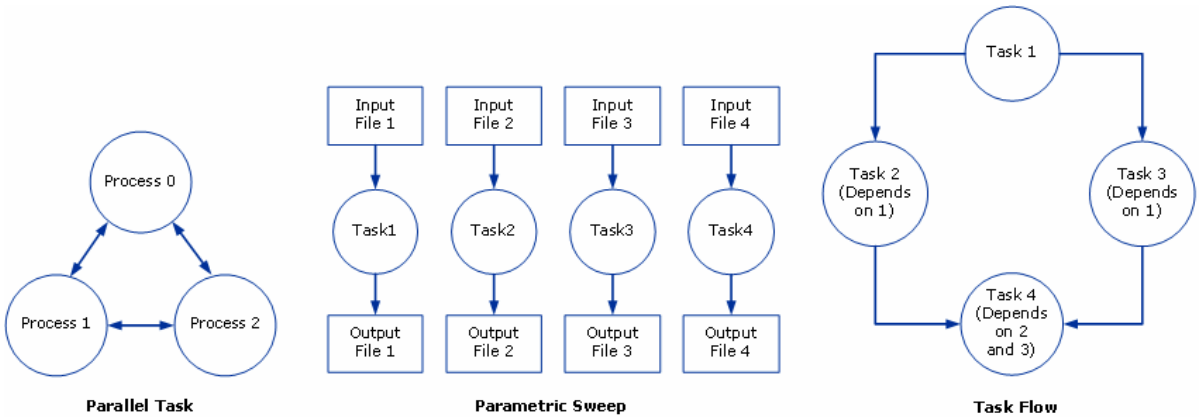


Figura 11. Tipos de Trabalho

Tipos de Trabalho

Existem três tipos de trabalhos suportados pelo WCCS, ilustrados na Figura 11:

- **Tarefa Paralela** (*Parallel Task*): conjunto de processos que executam simultâneamente trocando mensagens com o fim de cooperar para realizar alguma computação. Trata-se de um programa MPI, suportado pelo MSMPI no WCCS. Este tipo de trabalho é característico de cálculo científico e de engenharia;
- **Varredura Paramétrica** (*Parametric Sweep*): Numa varredura paralela, há várias tarefas independentes, que não se comunicam, sendo dispensável o ligamento a uma biblioteca MPI. Normalmente, cada tarefa toma sua entrada e provê sua saída em arquivos separados;
- **Fluxo de Tarefa** *(Task Flow)*: Conjunto de tarefas, geralmente distintas e independentes, que são executadas em uma ordem prescrita devido à interdependência de dados entre estas. Tarefas que executam simultaneamente, por não possuírem dependências de dados, como as tarefas 2 e 3 da Figura 11, não se comunicam, sendo o MPI também dispensável neste caso. O uso de *workflows* é bastante comum em aplicações de simulação, existindo *frameworks computacionais* especialmente desenvolvidos sob este paradigma.

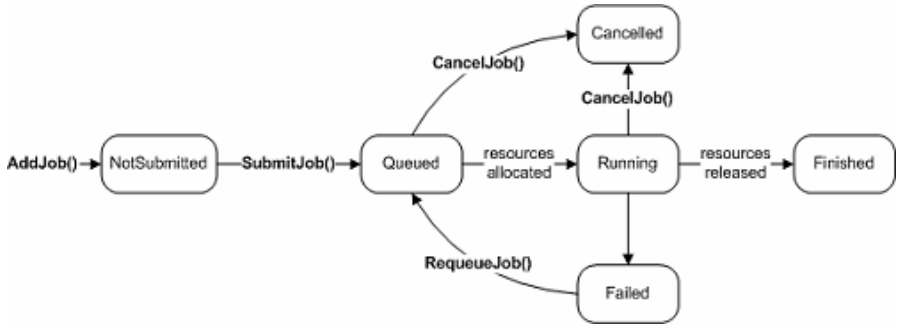


Figura 12. Ciclo de Vida de um Trabalho

Ciclo de Vida de Trabalhos e Tarefas

A Figura 12 e a Figura 13 apresentam os estágios do ciclo de vida de trabalhos e tarefas, os quais coincidem. No estágio NOT_SUBMITTED, um trabalho (tarefa) foi criado usando o comando `job new` (`job add`), mas ainda não submetido. A submissão ocorre com a execução do comando `job submit`, quando então o trabalho (tarefa) atinge o estágio QUEUED, aguardando ser executado, cancelado ou mesmo falhar. Neste último caso, quando o trabalho (ta-

refa) atinge o estágio FAILED, é permitido ao usuário resubmetê-lo, executando o comando `job requeue` (`task requeue`). Caso o usuário decida cancelar um trabalho (tarefa submetida), quando este encontra-se submetido (QUEUED) ou executando (RUNNING), usa-se o comando `job cancel` (`task cancel`). O trabalho (tarefa) é retirado da fila para execução de acordo com a política de escalonamento, quando atinge o estado RUNNING. Ao final da execução, os recursos alocados para o trabalho (tarefa) são liberados e este atinge o estado FINISHED.

Observe que na descrição acima, citamos vários dos operadores dos comandos job e task, mencionados anteriormente. Na verdade, estes controlam as intervenções do usuário no ciclo de vida dos trabalhos e tarefas. Além dos operadores citados, temos ainda os seguintes operadores para o comando `job`: `list`, o qual lista os trabalhos em execução; `listtasks`, o qual lista as tarefas contidas em um determinado trabalho; `modify`, para modificar um trabalho submetido ou em execução; e `view`, o qual mostra detalhes sobre um trabalho especificados. Para o comando `task`, temos ainda o operador `view`, o qual mostra detalhes sobre uma tarefa.

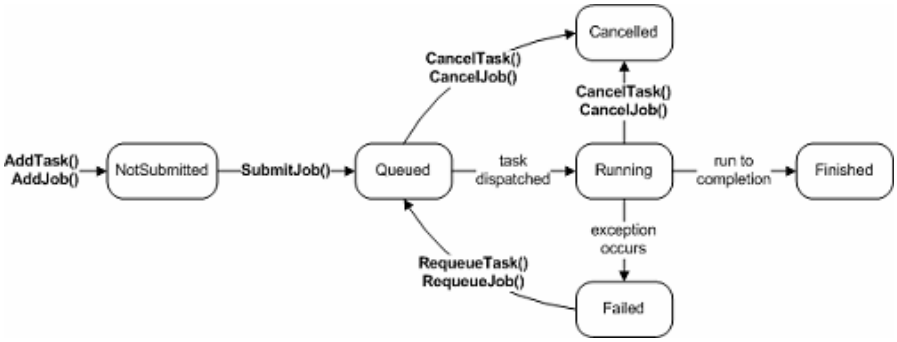


Figura 13. Ciclo de Vida de uma Tarefa

A Execução de Tarefas

É importante discutir como as tarefas de um trabalho são lançadas nos nós de computação de um *cluster*. O despacho de uma tarefa é feito pelo escalonador de trabalho, o qual o envia para um dos nós dentre aqueles designados para o trabalho. A menos que dependências entre as tarefas sejam definidas, estas são servidas segundo a política FCFS (os mais antigos são servidos primeiro). No caso de várias **tarefas sequenciais** (*varredura paralela* e *fluxo de tarefas*), estas são alocadas nó a nó, até que os processadores de cada nó estejam todos ocupados. Assim, se o nó designado pelo escalonador de trabalho dispõe de quatro processadores, o máximo suportado pela versão atual do WCCS, as quatro primeiras tarefas serão alocadas nestes quatro. As quatro seguintes, nos quatro processadores do próximo nó designado, até que todos os processadores, em todos os nós, estejam ocupados. Caso haja mais tarefas que processadores, estas devem aguardar a liberação de um processador. No caso de uma **tarefa paralela** que usa MSMPI, o executável `mpiexec` é executado no nó designado pelo escalonador de trabalho, disparando os processos que compõem a tarefa nos processadores do nó. Caso haja mais processos que processadores, vários nós podem ser utilizados. Para isso, uma instância do MSMPI deve ser disparada em vários nós antes mesmo que as tarefas sejam executadas.

Políticas de Escalonamento

A escolha de trabalhos a serem executadas pelo escalonador de trabalhos obedece a uma das seguintes políticas:

- ***Priority-based, first-come, first-served scheduling***: A fila de trabalho é composta por grupos de prioridade, também filas. Um trabalho é adicionado ao final da fila do grupo correspondente a prioridade a ele atribuída;
- ***Backfilling***: Permite que trabalhos menores executem antes que o trabalho que se encontra na frente da fila, de forma que este não tenha sua execução atrasada. Quando um trabalho alcança a frente da fila, podem não haver recursos suficientes para sua execução imediata. Em *backfilling*, os recursos disponíveis são alocados, porém utilizados pelo escalonador de trabalho para executar o primeiro trabalho na fila que possa ser executado imediatamente usando os recursos disponíveis e em tempo inferior ao tempo estimado para a(s) tarefa(s) em execução completarem. Dessa forma, o atraso ao trabalho que está de fato na frente da fila e a ociosidade dos recursos de processamento são minimizados;
- ***Exclusive scheduling***: caso um trabalho não utilize todos os processadores de um nó, estes ficam ociosos e indisponíveis a outros trabalhos. Porém, é possível desabilitar a exclusividade do acesso aos nós pelos trabalhos, recurso que deve ser usado com cuidado.

5. Ferramentas de Programação sobre WCCS

O *Compute Cluster Pack* (CCP) SDK tem sido desenvolvido para suporte a programação paralela sobre WCCS através dos ambientes de programação para ambiente Windows, como o Visual Studio, usado neste tutorial. Inclui um conjunto de bibliotecas e arquivos de cabeçalho para gerenciamento do *cluster*, interface com o gerenciador de trabalhos, e para construção de programas MPI e OpenMP. As duas primeiras correspondem ao suporte ao acesso ao componente CCAPI.DLL, cujas funcionalidades coincidem com aquelas já discutidas na seção anterior. De fato, esta seção concentra-se no uso do Visual Studio 2005, ambiente de programação padrão da Microsoft, para desenvolvimento de programas paralelos para execução em *cluster* WCCS, como trabalhos e tarefas. Para isso, apresentaremos exemplos de trabalho que exploram funcionalidades importantes, em especial a definição de tarefas dos tipos suportados pelo WCCS: *varredura paramétrica*, *fluxo de tarefas* e *tarefas paralelas*. Somente tarefas deste último tipo fazem uso do MPI. Portanto, nosso exemplo conterá um exemplo de tarefa paralela que ilustrará o uso do MSMPI. O uso do OpenMP será ilustrado através de um outro exemplo.

5.1. Exemplo 1: Tipos de Tarefas e Programação MPI

O objetivo deste exercício é exemplificar a criação e submissão de um trabalho que ilustra grupos de tarefas dos tipos *varredura paramétrica*, *fluxo de tarefas*, e *tarefa paralela MPI*. Com esta finalidade, implementamos três tarefas para:

1. Geração de matrizes;
2. Transposição de matrizes;
3. Multiplicação paralela de matrizes;

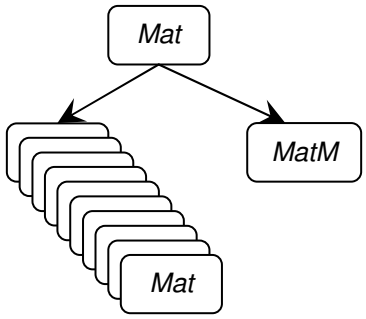


Figura 14. Fluxo de Tarefas

A Figura 14 ilustra o *fluxo de tarefas* sobre os quais as tarefas acima citadas estão organizadas. De fato, as tarefas *MatTrans* (2) e *MatMult* (3) dependem de matrizes produzidas pela tarefa *MatGen* (1), as quais são armazenadas em arquivos.

A Figura 16 apresenta uma captura de tela do ambiente do Visual Studio 2005, onde se vê o código de *MatGen*, o programa de geração de matrizes. O programa *matrix-generator.cpp* gera 10 matrizes de dimensão 3×3 preenchidas com números aleatórios. Não requer argumentos de entrada e fornece como saída a criação de 10 arquivos, com nomes `left<#>`, onde `<#>` varia entre 0 e 9, portanto `left0`, ..., `left9`. Os arquivos gerados por *MatGen* servirão de entrada para *MatTrans* e *MatMult*.

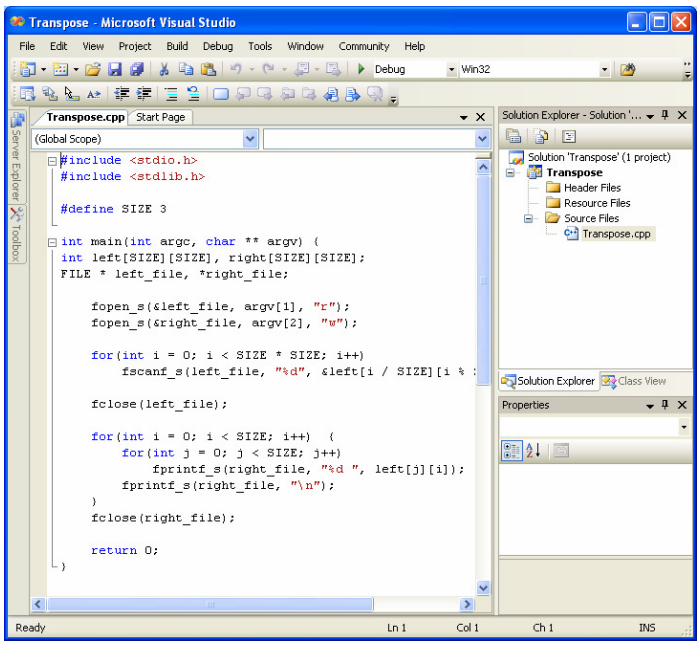


Figura 15. *MatTrans* – Transposição de Matrizes

A Figura 15 apresenta uma captura de tela do ambiente do Visual Studio 2005, onde se encontra aberta a solução *MatTrans*, referente ao programa de transposição de matrizes. Como é possível observar pelo código apresentado, o programa lê um arquivo onde está armazenada uma matriz de entrada. Então, a transpõe e o resultado é armazenado em um novo arquivo. Os nomes dos arquivos de entrada e saída são os argumentos do executável. Por exemplo,

```
Transpose.exe left0 transpose0
```

executa o programa `Transpose.exe`, de tal forma que este busque a matriz de entrada no arquivo `left0` e produza a saída no arquivo `transpose0`. De fato, o programa `Transpose.exe` será executado para cada arquivo de entrada produzido pelo programa de geração de matrizes, descritos anteriormente, formando uma *varredura paramétrica*. Portanto, note que em um *fluxo de tarefas*, as tarefas podem ser tanto sequenciais como varreduras paramétricas. Veremos pelo próximo programa, que uma tarefa em um *fluxo de tarefas* pode também ser uma *tarefa paralela MPI*.

A solução *MatMult*, apresentada na Figura 17, destina-se a realizar uma multiplicação paralela de matrizes de dimensão 3×3. Seu objetivo é apresentar um exemplo de programa MPI, representando uma tarefa paralela. Atualmente, as matrizes de entrada, chamadas *left* (esquerda) e *right* (direita), são geradas aleatoriamente pelo processo MATGEN, mas facilmente poderiam ser lidas de arquivos produzidos por *MatGen*, justificando o fluxo de tarefas da Figura 14. Três processos trabalhadores realizam a multiplicação das linhas da matriz *left* por uma das

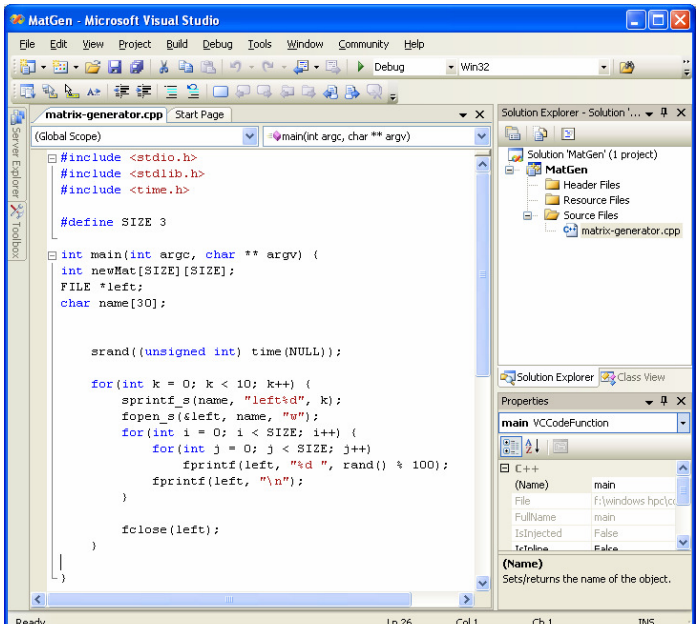


Figura 16. Solução *MatGen* - Geração de Matrizes

colunas da matriz *right* (rotina *multiplyRowCol*). Para isso, o processo ROOT envia a matriz *left* e uma coluna da matriz *right* para cada um dos três processos trabalhadores envolvidos (rotina *invokeMultiply*). O processo MATPRINT é responsável por escrever a matriz resultante no dispositivo de saída. Portanto, são necessários 6 processos, cujo papel é definido pelo seu *rank* (identificador do processo). Usaremos este programa para ilustrar as principais características de programas MPI.

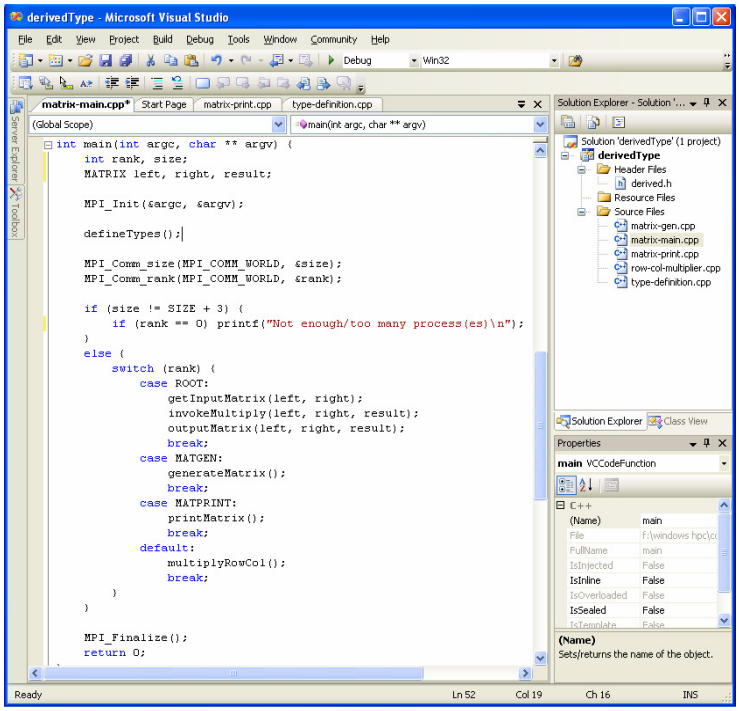


Figura 17. *MatMult* - Multiplicação de Matrizes

O código apresentado na Figura 17 mostra o corpo da função principal do programa. Trata-se de um programa paralelo que segue o padrão SPMD (*Single Program Multiple Data*), típico de programas MPI. Dessa forma, o programa executável, denominado `derivedType.exe`, é disparado cada um entre seis processadores, sendo atribuído um *rank* (número inteiro) diferente a cada cópia. O *rank* corresponde à identificação do processo. Utilizando o MSMPI, o disparo de um programa paralelo em seis processadores deve ser feito utilizando o comando `mpiexec`, a partir de um *prompt* de comando no nó principal do *cluster*. Para o exemplo mencionado, eis um exemplo:

```
mpiexec -n 6 derivedType.exe
```

As rotinas `MPI_Init`, `MPI_Comm_size`, e `MPI_Comm_rank` inicializam o ambiente MPI necessário para execução do programa. A rotina `MPI_Init` inicializa o ambiente MPI, recebendo os argumentos passados pelo comando `mpiexec` ao programa executável. A rotina `MPI_Comm_size` permite ao programa conhecer o número de processos em execução, o qual é, no exemplo, armazenado na variável *size*. Esta informação é particularmente útil quando um programa pode ser disparado em um número arbitrário de processadores, de forma a parametrizar dinamicamente o programa usando esta informação. No exemplo, onde o número de processos é fixo e igual a seis, o número de processos é usado para impedir que um usuário inadvertidamente execute `derivedType.exe` com um número diferente de 6 processos A rotina `MPI_Comm_rank` informa ao processo o seu *rank*, o armazenado na variável *rank*. No final do programa, a rotina `MPI_Finalize` deve ser chamada para liberar os recursos do ambiente paralelo que estão sendo alocados pelo ambiente MPI para o programa. O não uso desta rotina pode causar efeitos imprevisíveis. Entre a inicialização e finalização, os processos envolvidos em um programa MPI podem cooperar para realizar uma tarefa comum, através da troca de dados, viabilizado por um conjunto de subrotinas que habilitam a passagem de mensagens entre processos.

O MPI suporta dois conjuntos de rotinas de trocas de mensagens: *ponto-a-ponto* e *coletivas*. As primeiras permitem copiar o conteúdo de um *buffer* de dados de um processo fonte para um outro *buffer* em um processo destino. As rotinas de envio ponto-a-ponto básicas, em C, são MPI_SEND e MPI_RECV, com as quais virtualmente qualquer programa paralelo pode ser construído. Há outras versões para MPI_SEND, as quais mudam a semântica do envio e permitem obter melhor eficiência em situações específicas: MPI_BSEND (*bufferizado*), onde o processo fonte não é bloqueado a espera da chamada MPI_RECV correspondente; MPI_RSEND (*ready*), o

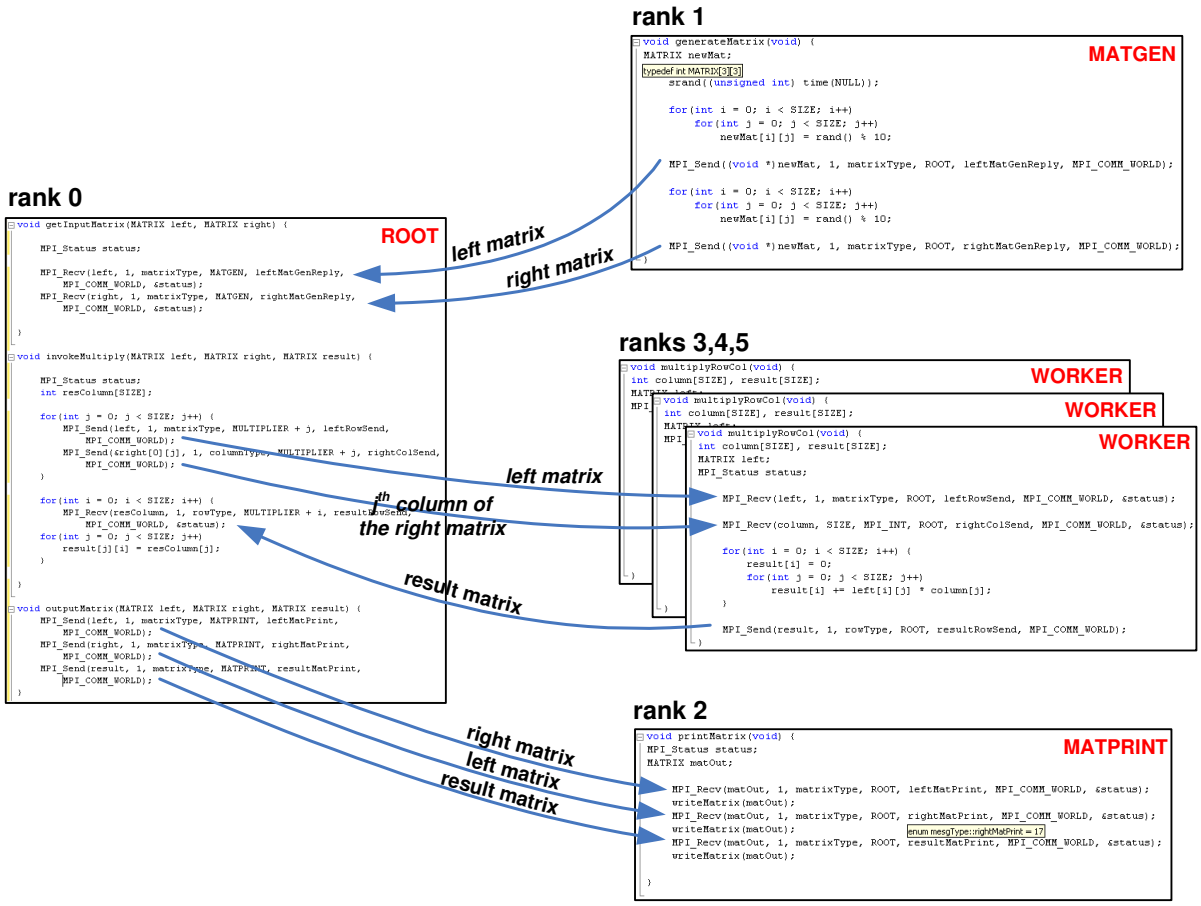
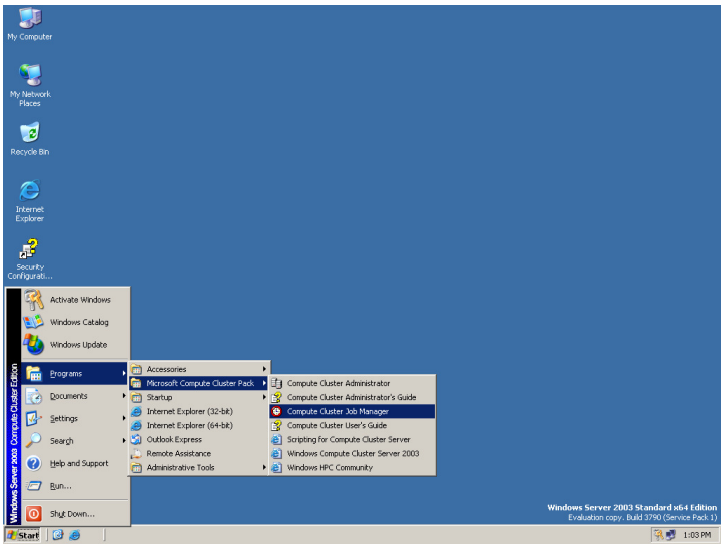


Figura 18. Trocas de Mensagens em *MatMult*

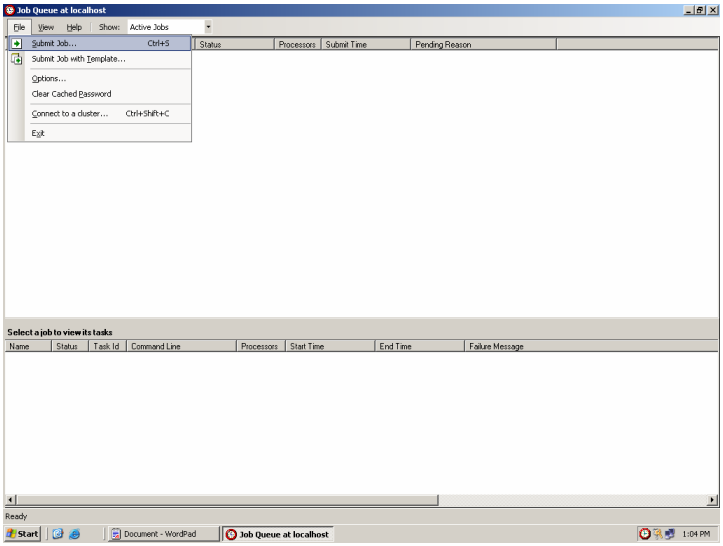
qual obriga que a chamada MPI_RECV correspondente tenha sido realizada anteriormente; e MPI_SSEND (*síncrono*), o qual força a parada do processo fonte a espera da chamada MPI_RECV correspondente. O MPI_SEND pode comportar-se ora como MPI_SSEND, ora como MPI_BSEND, dependendo da implementação do MPI utilizada. Há ainda as versões assíncronas para as rotinas ponto-a-ponto (MPI_ISEND, MPI_IRSEND, MPI_IBSEND, MPI_ISSEND, e MPI_IRECV), as quais permitem sobrepôr comunicação e computação com a finalidade de reduzir o tempo de ociosidade dos processadores. As rotinas de comunicação coletiva permitem a troca de dados entre um conjunto de processos envolvidos, os quais atuam como pares (*peers*), segundo vários padrões de trocas de dados capturados pela semântica das subrotinas MPI_BCAST, MPI_GATHER, MPI_SCATTER, MPI_REDUCE, MPI_ALLGATHER, MPI_ALLTOALL, MPI_ALLREDUCE, e MPI_SCAN. MPI suporta ainda *escopos de comunicação*, tipos *de dados derivados*, *topologias*, e *processos dinâmicos*. Sugerimos ao leitor consultar a documentação do MPI para detalhes.

O exemplo da Figura 18 é completamente baseado em chamadas a rotinas ponto-a-ponto. Como se pode observar, o papel de cada processo (ROOT, MATGEN, MATPRINT ou trabalhador) é definido pelo seu *rank*. Dependendo deste, a sequência correta de chamadas de rotinas é efetivada. Por exemplo, o processo ROOT (*rank* 0), executa em sequência as seguintes sub-rotinas: *getInputMatrix*, para receber as matrizes geradas pelo processo MATGEN; *invokeMultiply*, onde as matrizes são distribuídas e enviadas aos processos trabalhadores, e onde os resultados parciais são recebidos destes, sendo agrupados em uma única matriz; e *outputMatrix*, a qual envia as matrizes de entrada (*left* e *right*) matriz de saída (*result*) ao processo MATPRINT, responsável por apresentá-las no dispositivo de saída. As trocas de mensagens entre os processos envolvidos no programa *MatMult* são ilustrados na Figura 18. A seguir, apresentamos passos para submeter o trabalho apresentado neste exemplo para execução no WCCS.

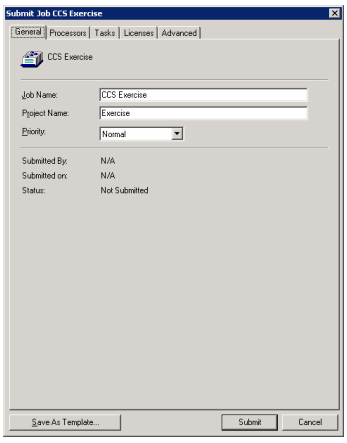
1. Inicie o *Compute Cluster Job Manager* a partir do menu Iniciar.



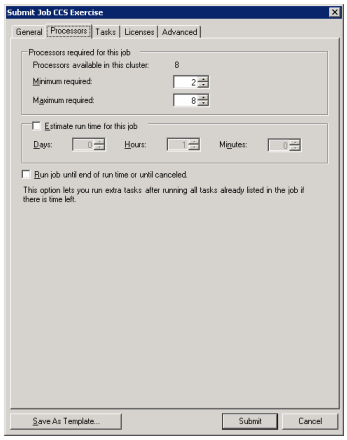
2. Clique **File–>Submit Job**



3. Entre os nomes do trabalho (**Job Name**) e do projeto (**Project Name**).

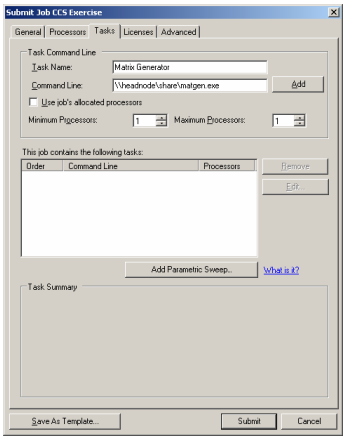


4. Selecione o mínimo e máximo número de processadores requeridos para o trabalho. Selecione o mínimo de 2 e máximo 8 para o exercício.

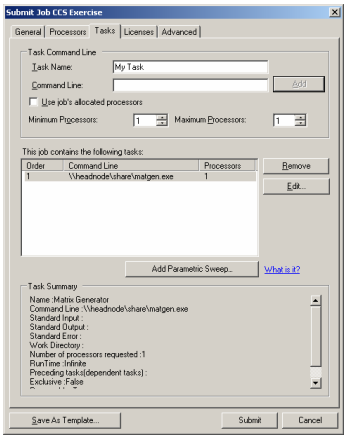


5. Copie os executáveis em C:\Exercise\Module 5\ over to \\headnode\share de forma que estes fiquem disponíveis para todos os nós de computação, sem que estes sejam fisicamente copiados para os nós.

6. As tarefas individuais que constituem o trabalho devem ser configuradas. Crie uma nova tarefa especificando o nome desta como **Matrix Generator,** preenchendo **Command Line** com **\\headnode\share\MatGen.exe** na aba de tarefas. Então, clique **Add.**

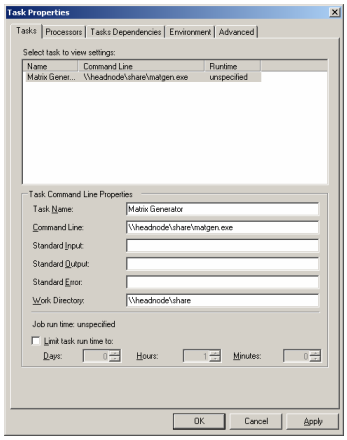


7. Você verá a tarefa aparecer na janela de trabalho, como mostrado ao lado**.**



8. Clique **Edit** para abrir **Task Properties** (propriedades da tarefa), após selecionar as novas tarefas.

9. Especifique o caminho do **Work Directory**, onde arquivos temporários e de saída devem ser armazenados. Note que este caminho deve estar acessível por todos os nós que executma o trabalho. Atualize o diretório de trabalho para **\\headnode\share**



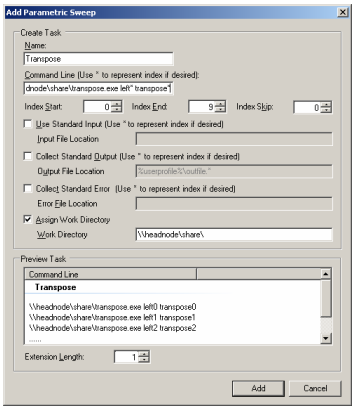
10. Clique em **Add Parametric Sweep** para definir a Tarefa 2.

11. Dê um nome apropriado para a Tarefa 2 (**Transpose)**

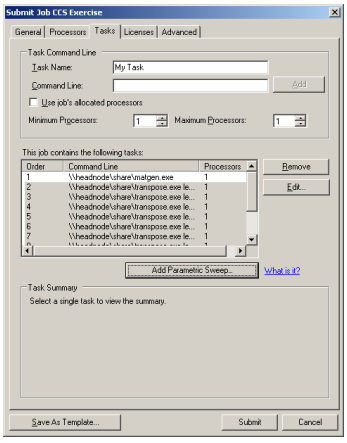
12. Em **Command Line,** especifique o caminho completo do comando **Transpose left* transpose*** (* substituído pelos índices nas iterações)

13 Configure **Index Start** como **"0", Index End** como **"9"** e **Index Skip** como **"0"** para a décima iteração da Tarefa 2.

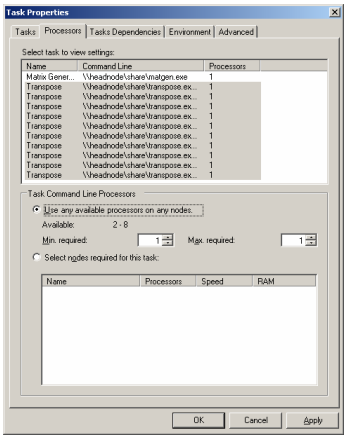
14 Verifique a caixa **Assign Work Directory** e especifique o caminho onde arquivos temporários e de saída devem ser armazenados. Este caminho deve estar acessível por todos os nós que executam o trabalho. Atualize o diretório de trabalho para **\\headnode\share.**



15. Clique **Add**. As iterações para a Tarefa 2 aparecerão, como mostrado abaixo.

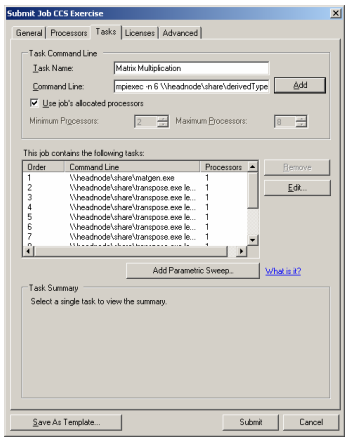


16. Clique **Edit** após selecionar todas as tarefas relacionadas à varredura paramétrica, para atribuir processadores a tarefa

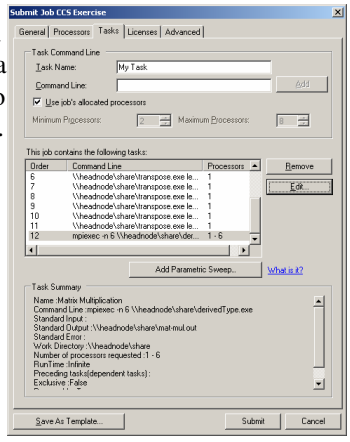


17. Configure o mínimo e máximo requeridos de processadores para 1e clique **OK**

18. Você já pode submeter a tarefa 3. Na linha de comando, digite **mpiexec –n 6 \\headnode\share\derivedType.exe** e clique **Add.**



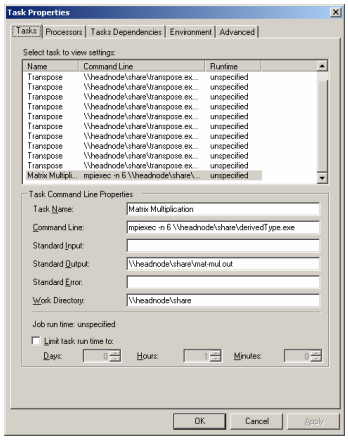
19. Você verá a tarefa aparecer na janela de trabalho, como mostrado ao lado.



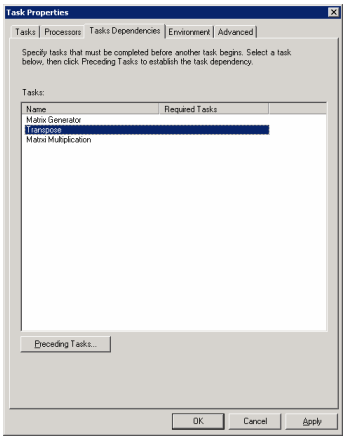
20. Clique **Edit** para abrir **Task Properties**

21. No campo **Standard Output** dê um nome de arquivo de saída, com extensão.

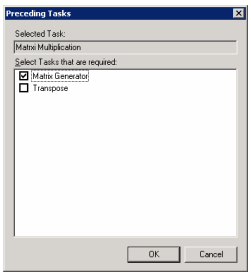
22. Especifique o caminho do **Work Directory** (diretório de trabalho), onde arquivos temporários e outros arquivos de saída serão armazenados. Note que este caminho deve ser acessível por todos os nós que executam o trabalho. Atualize o diretório de trabalho para **\\headnode\share**



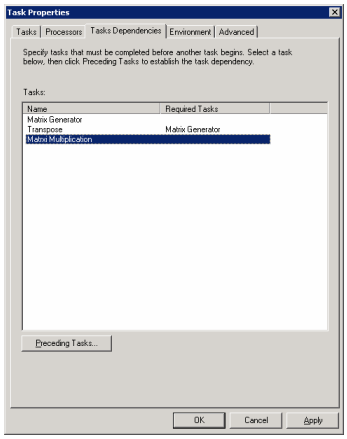
23. Selecione a aba **Task dependencies**, selecione a tarefa **Transpose,** e clique em **Preceding Tasks.**



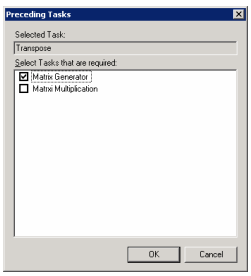
24. Na janela de precedência de tarefas, faça a tarefa **Matrix Generator** para preceder a tarefa **Transpose** e clique **OK**



25. Na aba **Task dependencies**, selecione a tarefa **Matrix Multiplication** e clique em **Preceding Tasks**

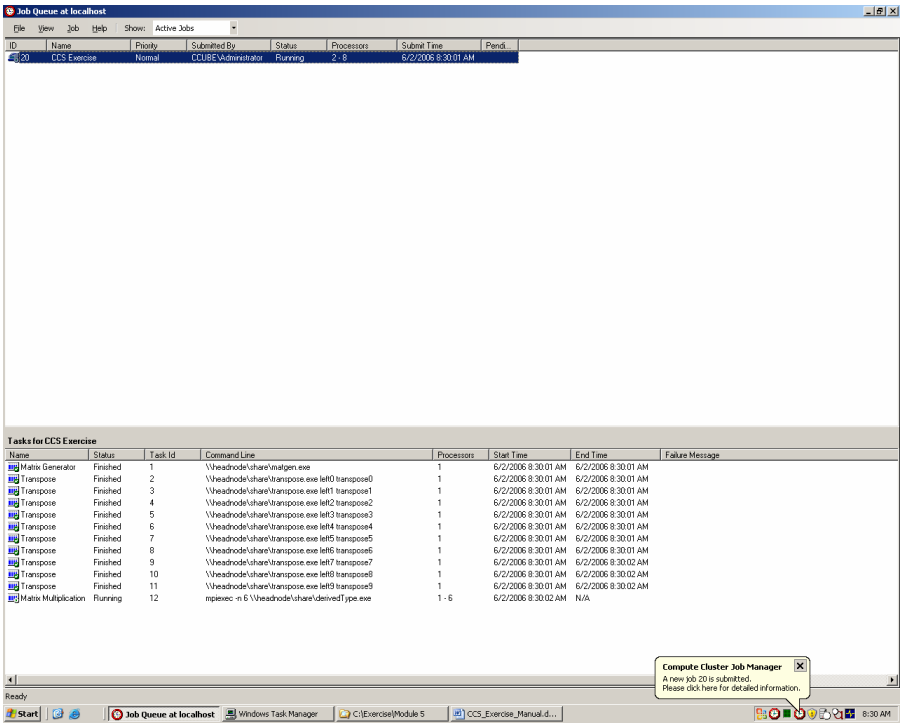


26. Na janela de precedência de tarefas, faça a tarefa **Matrix Generator** preceder a tarefa **Matrix Multiplication** e clique **OK**



27. Configurada a dependência entre as tarefas, clique **OK** para salvar as alterações e então clique em **Submit** na janela de submissão de trabalhos.

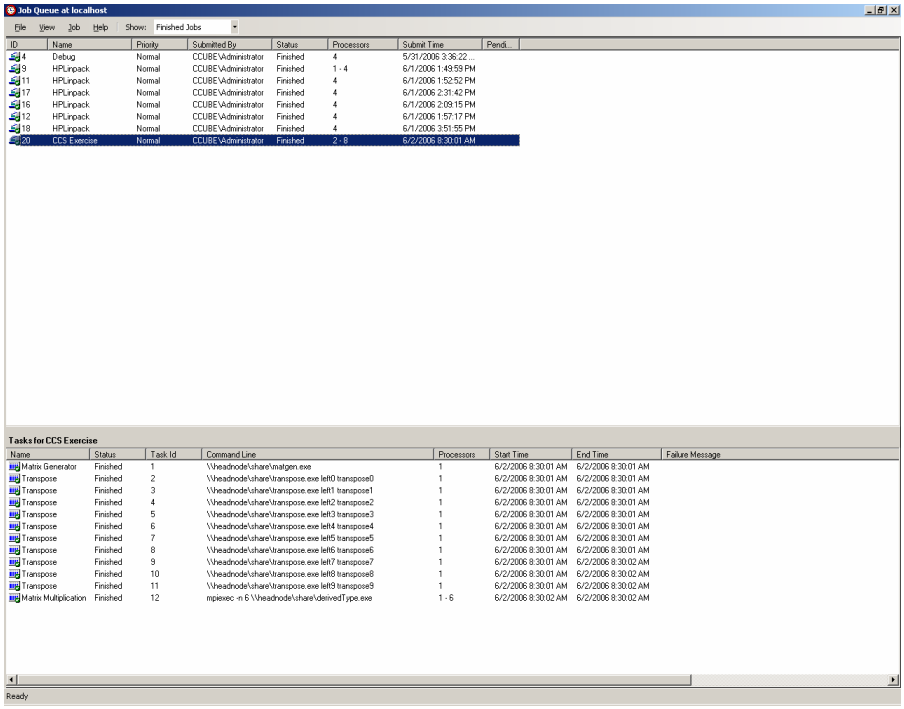
28. Os trabalhos surgirão na janela de trabalhos ativos (**Active Jobs**). Uma vez terminados você poderá vê-los na janela de trabalhos finalizados (**Finished Jobs**)



29. Verifique se os arquivos left0, left1 … left9, cada um contendo uma matriz de dimensões 3×3, foi criado em **\\headnode\share**

30. Verifique se os arquivos transpose0, transpose1 … transpose9, cada um contendo as transpostas das matrizes em left0, left1 … left9 respectivamente, estão em \\headnode\share.

31. O arquivo \\headnode\share\mat-mul.out deve conter as matrizes left e right, geradas randomicamente, e seu produto após o trabalho completar-se com sucesso.



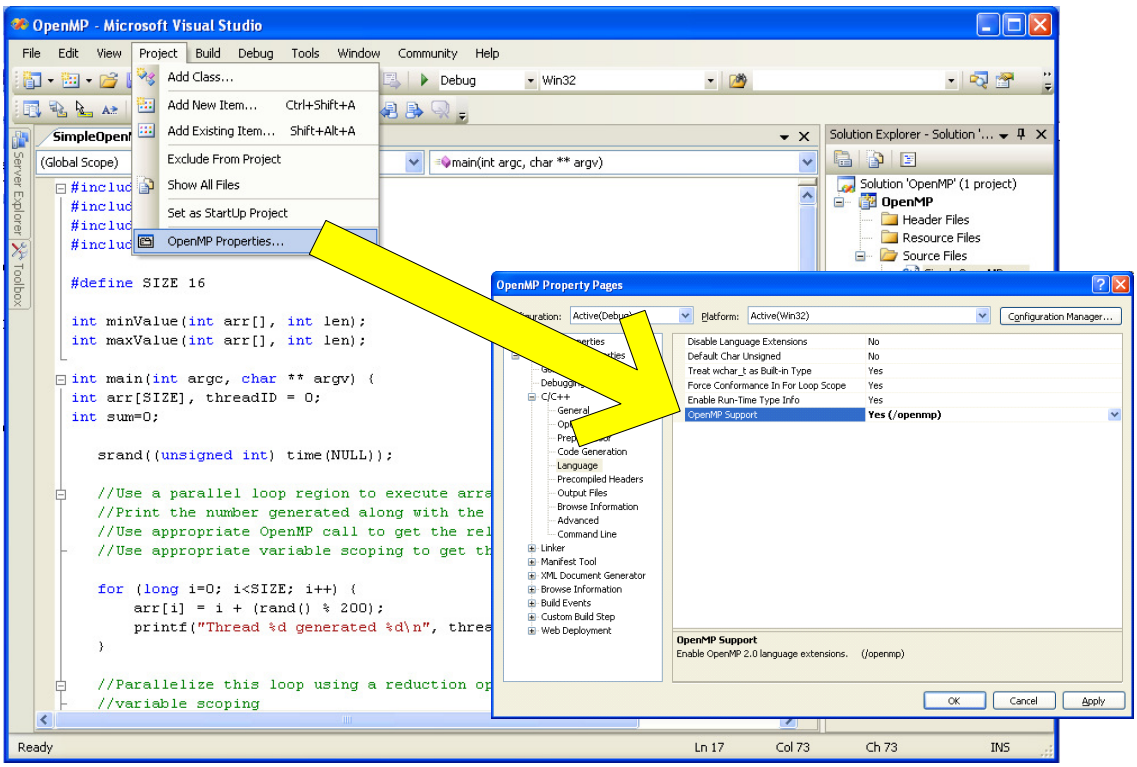


Figura 19. Habilitando o OpenMP em um Projeto

5.2. Exemplo 2: OpenMP

O OpenMP acrescenta a linguagens de programação um conjunto de diretivas que permitem executar iterações de um laço e trechos de programa de maneira concorrente, supondo memória compartilhada entre o conjunto de *threads*. O exemplo apresentado a seguir tem por objetivo mostrar os principais aspectos da estrutura de um programa OpenMP.

A Figura 19 ilustra com deve-se habilitar o suporte a OpenMP em um projeto C/C++ no Visual Studio. Uma vez habilitado, o programa executável é capaz de gerar e gerenciar um conjunto de *threads* que executam sobre os processadores de um nó do *cluster*. O programa `SimpleOpenMP.cpp` é um programa sequencial que gera um vetor de tamanho SIZE (trecho 1), soma os elementos do vetor (trecho 2) e imprime o maior e menor elemento do vetor (trecho 3). Neste, introduziremos diretivas OpenMP com a finalidade de executar em paralelo estes trechos, ilustrando diferentes formas de paralelização suportada por OpenMP:

Trecho 1

O código atual inicializa um vetor *arr* de SIZE elementos com número aleatórios:

```
for (long i=0; i<SIZE; i++)
{
    arr[i] = i + (rand() % 200);
    printf("Thread %d generated %d\n", threadID, arr[i]);
}
```

Desejamos paralelizar o código acima, de forma que uma *thread* é responsável pela inicialização de cada elemento do vetor *arr*. Para isso, devemos incluir diretivas OpenMP:

```
#pragma omp parallel shared(arr) private(threadID)
{
    threadID = omp_get_thread_num();
    #pragma omp for
    for (long i=0; i<SIZE; i++)
    {
        arr[i] = i + (rand() % 200);
        printf("Thread %d generated %d\n", threadID, arr[i]);
    }
}
```

A diretiva **`#pragma omp parallel`** indica ao compilador que a seção de código delimitada que se segue é uma *região paralela*, onde *threads* concorrentes executarão, sendo o vetor *arr* compartilhado entre as *threads* e a variável *threadID* local a cada *thread*. A diretiva **`#pragma omp for`** então instrui o compilador a executar cada iteração do laço **for** em uma *thread*.

Trecho 2

O código do trecho 2 soma os elementos do vetor *arr*:

```
for (long i=0; i<SIZE; i++)
{
      sum += arr[i];
}
```

Desejamos paralelizar o trecho acima, de tal forma que uma *thread* é responsável pela soma de um subconjunto dos elementos do vetor *arr*. Para isso, devemos incluir seguintes diretivas OpenMP:

```
#pragma omp parallel shared(arr) private(threadID) reduction (+:sum)
{
    threadID = omp_get_thread_num();
    #pragma omp for
    for (long i=0; i<SIZE; i++)
    {
        sum += arr[i];
    }
    printf("Thread %d sum = %d\n", threadID, sum);
}
```

A cláusula **`reduction (+:sum)`** instrui o compilador a considerar a variável *sum* privada a cada *thread* e somar os valores locais destas ao final da barreira da região paralela. Portanto, no código acima, cada *thread* imprimirá o valor local calculado por cada *thread*.

Trecho 3

O código do trecho 3 imprime o maior e menor elemento do vetor *arr*, sequencialmente:

```
printf("Minimum value = %d, computed by thread %d\n",
                minValue(arr, SIZE), threadID);
printf("Maximum value = %d, computed by thread %d\n",
                maxValue(arr, SIZE), threadID);
```

Desejamos paralelizar o trecho acima, de forma que o cálculo do menor e maior valor sejam realizados em paralelo. São necessárias exatamente 2 *threads*. Para isso, devemos incluir seguintes diretivas OpenMP para criação de *seções paralelas* separadas pela diretiva `#pragma omp section`:

```
#pragma omp sections num_threads(2)
{
   printf("Minimum value = %d, computed by thread %d\n", minValue(arr, SIZE),
                          omp_get_thread_num());
   #pragma omp section
   printf("Maximum value = %d, computed by thread %d\n", maxValue(arr, SIZE),
                          omp_get_thread_num());
}
```

7. Maiores Informações

O principal portal mantido pela Microsoft para acesso a informações sobre o *Windows Compute Cluster Server* é http://www.microsoft.com/windowsserver2003/ccs/. A partir deste, temos acesso aos vários programas de parceria da Microsoft com outras indústrias da área de computação de alto desempenho, estudos de caso da aplicação do WCCS na indústria[14], *download* de atualizações, avaliação de produtos, conferências on-line (*webcasts*), etc. É ainda possível acessar informações técnicas detalhadas sobre o WCCS, mantidas publicamente em vários artigos (*white papers*) e através do portal *Microsoft Windows Server TechNet*, http://technet2.microsoft.com/windowsserver/en/technologies/featured/ccs/default.mspx. Já o portal *Windows HPC Community* (http://windowshpc.net/) é dedicado a comunidade de usuários do WCCS, também provendo várias informações técnicas úteis. Com finalidade semelhante, há o *newsgroups* **microsoft.public.windows.hpc,** o qual pode ser acessado a partir do endereço http://www.microsoft.com/communities/newsgroups/list/en-us/default.aspx. A Microsoft mantém ainda colaboração com diversas universidades e centros de pesquisa ao redor do mundo através do programa *Microsoft HPC Institute*. Atualmente são dez centros, espalhados na Ásia, América do Norte e Europa. Maiores informações podem ser obtidas em http://msinst-home.spaces.live.com/. No Brasil, devemos citar o Laboratório Microsoft na UNICAMP, o qual tem produzido material relacionado ao WCCS, disponível no site http://codeplex.com/NDOS (NDOS-BR, ou Núcleo de Desenvolvimento Open Source e Interoperabilidade), mantido pela Microsoft para a comunidade, onde estão abrigado projetos brasileiros de código livre em geral relacionados a interoperabilidade entre soluções Linux e Windows.

Há ainda portais independentes, como http://www.winhpc.org (*WinHPC.org - Windows HPC and Computer Cluster Server Portal*), mantido por Ken Farmer em http://www.spyderbyte.com/. Neste, são oferecidas informações sobre novidades recentes da indústria, listas de *e-mail*, e *links* para outras fontes de informações relativas a computação de alto desempenho no ambiente Windows.

[14] http://www.microsoft.com/casestudies/search.aspx?Keywords=compute+cluster+server.

7. Considerações Finais

As técnicas de computação de alto desempenho, notadamente aquelas que tornam possível coordenar eficientemente vários processadores para executar um *software* computacionalmente intensivo, têm tido um importante papel para viabilizar um grande número de aplicações de interesse de instituições acadêmico-científicas e indústrias, notadamente oriundas das ciências computacionais, engenharia, e, mais recentemente, finanças. Tradicionalmente, os avanços nesta área têm sido alavancados devido ao interesse da indústria do *hardware*, em cooperação com instituições de pesquisa e universidades, resultando em um conhecido fosso entre o desempenho do *hardware* aplicado a computação de alto desempenho e a capacidade do *software* de explorá-lo de forma compatível com os princípios da engenharia de *software*.

O WCCS é ainda um passo inicial, porém uma alternativa de uso prático efetivo para viabilizar um ambiente efetivo de computação paralela, uma vez que utiliza interfaces de programação já popularizadas em plataformas Linux, como MPI e OpenMP. Entretanto, um longo caminho há ainda de ser percorrido para a consolidação de interfaces de mais alto nível que integrem de fato a programação paralela às plataformas modernas de desenvolvimento de *software*, adaptadas à perspectiva da disciplina de engenharia de *software*, tornando a programação paralela mais acessível a seus usuários e menos exigente com relação aos conhecimentos que estes deve possuir sobre características peculiares das arquiteturas e como estas podem ser exploradas. Os requisitos que tornam difícil esta integração são, principalmente, a necessidade de estas ferramentas manterem o mesmo nível de desempenho de MPI, como usado atualmente, e possuírem expressividade suficiente para comportar os diversos padrões de paralelismo comuns em aplicações de computação de alto desempenho. Neste contexto, é importante a investigação de como incorporar a execução eficiente de programas paralelos em ambientes de execução gerenciada, como .NET e JVM, cujo uso tem proliferado na indústria. A plataforma .NET parece incorporar várias características importantes para computação de alto desempenho, como o suporte a múltiplas linguagens, a portabilidade de código objeto, o suporte a execução de código não-gerenciado, e a existência de um sistema de tipos incorporado ao código intermediário, o qual poderia ser enriquecido com tipos de dados típicos de aplicações de alto desempenho.

É importante ainda ressaltar que os requisitos de ambientes de computação de alto desempenho extrapolam a necessidade de boas ferramentas de programação paralela. Isso ficou evidenciado ao identificarmos requisitos comuns e relevantes de aplicações contemporâneas de alto desempenho na Seção 2, em especial com relação aos mecanismos de acesso a extensas bases de dados que não são viáveis de serem copiadas localmente para processamento e a integração de modelos em ambiente de grade computacional, cada qual possivelmente representado por um programa paralelo que executa em um *cluster* e que pode ser acessado remotamente por meio de interface de serviço. Este é o paradigma conhecido como programação paralela distribuída.

Portanto, espera-se que o interesse da indústria do *software* por aplicações de alto desempenho, bastante influenciado pelas recentes iniciativas da Microsoft, venha a alavancar a evolução das tecnologias de *software* para computação de alto desempenho. Este tutorial apresentou então os principais aspectos sobre a principal iniciativa da Microsoft na área de alto desempenho, com o lançamento do *Windows Compute Cluster Server* (WCCS). Devido às premissas básicas de usabilidade que norteiam as soluções baseadas no ambiente Windows, O WCCS surge como uma alternativa que se propõe ser eficiente, simples, e de fácil implantação, integrando-se a outros *softwares* do ambiente Windows através de APIs de acesso às funcionalidades do *cluster* (CCAPI.DLL). Em especial, é destinado aos usuários da plataforma Windows que demandam de computação paralela, mas que consideram custosas as alternativas de migração da plataforma Windows, com a qual se encontram satisfeitos, para a plataforma Linux, ou a adoção de soluções heterogêneas.

Bibliografia

[1] BAKER, M. - BUYYA, R.- HYDE, D. (1999). *Cluster Computing: A High Performance Contender*. Communications of the ACM, vol.42, n.7, 79-83.

[2] BELL, G. - GRAY, J. (2002) *What's the Next in High Performance Computing*. Communications of the ACM, Vol. 45, n. 2, pp. 91-95.

[3] BERNHOLDT D. E. (2004) *Raising Level of Programming Abstraction in Scalable Programming Models*. Proceedings of IEEE International Conference on High Performance Computer Architecture (HPCA), Workshop on Productivity and Performance in High-End Computing (P-PHEC), pp. 76-84, Madrid, Spain.

[4] CARVALHO JUNIOR, F. H., LINS, R. D. (2007) *Towards an Architecture for Component-Oriented Parallel Programming*. Concurrency and Computation: Practice and Experience, v. 19 n. 6, pp. 697-719. DOI: 10.1002/cpe.1121.

[5] CARVALHO JUNIOR, F. H., LINS, R. D. (2005) *Separation of Concerns for Improving Practice of Parallel Programming*. Journal of Information and Computation, International Information Institute, v. 8, n. 5, Tokio, Japan.

[6] DONGARRA, J. – FOSTER, I. – FOX, G. – GROPP, W. – KENNEDY, K. – TORCZON, L. – WHITE (2003), A. *Sourcebook of Parallel Computing*. Morgan Kauffman Publishers.

[7] DONGARRA, J. (2004), A. *Trends in High Performance Computing*. The Computer Journal. Vol. 47, n.4, pp. 399-403.

[8] FOSTER, I. - KESSELMAN, C. (2004) *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kauffman Publishers.

[9] GEIST, G. A. – BEGUELIN, A. – DONGARRA, J. – JIANG, W. – MANCHEK, R. – SUNDERAM, V. S. (1994) *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge.

[11] MPI-FORUM. (1994) *MPI: A Message-Passing Interface Standard*. International Journal of Supercomputer Applications and High Performance Computing.

[12] OpenMP Architecture Review Board. OpenMP: *A Proposed Industry Standard API for Shared Memory Programming*. Relatótio Técnico, 1997.

[10] POST, D. E. – VOTTA, L. G. (2005), *Computational Science Demands a New Paradigm*, Physics Today, vol.58, n. 1, 35-41.

[13] VOGELS, W. (2003) *HPC.NET – are CLI-Based Virtual Machines Suitable for High Performance Computing*. Conference on High Performance Networkin and Computing, Proceedings of the 2003 ACM/IEEE Conference on Supercomputing.

[14] ROBSON, D. (2007) *Maths Maps Tomorrow's Drugs*. Scientific Computing World, issue 92, pp. 24-25, Warren Clark (editor). http://www.scientific-computing.com.

[15] GRANT, F. (2007) *Getting fisical with Phynance*. Scientific Computing World, issue 91, pp. 14-26, Warren Clark (editor). http://www.scientific-computing.com.

[16] BLACK, S. – SCHOLES, M. (1973) *The Price of Options and Corporate Liabilities*. Journal of Political Economy, 81(3): pp. 657-654.

[17] CARSON, E. – COBELLI, C. – BRONZINO, J. (editors) (2006) *Modelling Methodology for Phisiology and Medicine*. Academic Press Biomedical Engineering Series, ISBN 978-0121602451.

[18] SPENCER JR, B. ET AL (2004) *NEESGrid: A Distributed Collaboration for Advanced Earthquake Engineering Experiment and Simulation*. 13th World Conference on Earthquake Engineering, Vacouver, Canadá, Aug. 1-6, paper #1674.

[19] SHAYA, E. – THOMAS, B. (2001) *Specifics on a XML Data Format for Scientific Data*. Astronomical Data Analysis Software and Systems X, ASP Conference Series, v. 238, pp. 217-220.

[20] SKILLICORN, D. B. – TALIA, D. (1998) *Models and Languages for Parallel Computation*. ACM Computing Survreys, 30(2), pp. 123-169, ACM Press.

[21] BECKER, D. J. – STERLING, T. – SAVARESE, D. – DORBAN, J. E. – RANAWAK, U. A.. (1995) *Beowulf: A Parallel Workstation for Scientific Computation*. Proceedings of the 1995 International Conference on Parallel Processing.

[22] ANDERSON, T. E. – CULLER, D. E. – PATTERSON, D. A. (1994) *A Case for Networks of Workstations*. IEEE Euromicro, 15(1), pp. 54-64.

[23] SEYMOUR, K. – YARKHAN, A. – DONGARRA, J. (2005) *NetSolve: Grid Enabling Scientific Computing Environments*. Advances in Parallel Computing, v.14, Elsevier.

APÊNDICE A. Requisitos de *Hardware* do WCCS.

- **Processador:** Processadores Intel com suporte a arquitetura x64, como os da família Pentium e Xeon, equipados com suporte ao *Intel Extended Memory 64 Technology* (EM64T); Processadores AMD64, como os da família AMD Opteron, Athlon e compatíveis;
- **Memória RAM:** 512MB;
- **Multiprocessamento:** Até quatro processadores por nó;
- **BIOS:** Certifique-se que a versão da BIOS é a mais atual do fabricante;
- **Espaço mínimo de disco para instalação:** 4GB;
- **Volumes de disco:**
  - **Nó principal:** Configure dois volumes (C:\ and D:\) se planeja-se adicionar nós usando o método de adição automática (*Automated Addition method*). O primeiro será a partição do sistema, enquanto o outro será usado para armazenar as imagens do *Remote Installation Services* (RIS) necessárias ao método automático. A partição que armazena as imagens RIS devem conter no mínimo 2GB e formatadas usando NTFS. Suficiente espaço de disco deve existir para armazenar arquivos de entrada e saída de dados associados com aplicações e *scripts* que serão usados pelos usuários do *cluster*;
  - **Nó de Computação:** Um único volume de disco é geralmente suficiente;
- **Adaptadores de rede:** Todos os nós precisam de pelo menos um adaptador de rede. Se planeja-se utilizar uma uma rede privada, o nó principal requer pelo menos dois adaptadores de rede para criação das redes pública privada, dependendo da topologia de rede selecionada. Cada nó pode necessitar de adaptadores adicionais nos casos de acesso a rede pública ou suporte a uma rede MPI;
- **Suporte a *boot* PXE (BIOS e adaptador de rede):** Se você planeja o uso do método automático de adição de nós de computação, servidores que você planejá adicionar devem suportar PXE na sequência de *boot*. Isto é normalmente configurado na BIOS. Adicionalmente, verifique se o adaptador de rede de cada computador que é usado na rede privada suporta *boot* PXE.
- **Controladores 64-bit:** Verifique que você possui todos os controladores de suporte a modo 64 bits para o *hardware* dos nós. Se você pretende usar o método de adição automática de nós, certifique-se que os controladores necessários encontram-se na imagem RIS apropriada.