Sistema Gerenciador de Processamento Cooperativo

Ivonne Martínez Carrazana
Nelson C. Machado
Célio C. Guimarães

Relatório Técnico DCC–05/93

Abril de 1993

Sistema Gerenciador de Processamento Cooperativo

Ivonne Martínez Carrazana
Nelson C. Machado
Célio C. Guimarães*

Sumário

Este trabalho apresenta o projeto e a implementação de um Sistema Gerenciador de Processamento Cooperativo (SGPC). A utilização do SGPC facilita a exploração da capacidade de paralelismo inerente a uma rede local. Foram desenvolvidos um **Servidor de Processamento** e um pacote de rotinas que serve de interface para o servidor.

A utilização do sistema pode ser feita em um de dois níveis: seguindo o modelo Cliente/Servidor através de Chamadas Remotas de Procedimentos (RPCs) diretamente ao servidor, ou através de subrotinas de biblioteca que se encarregam de gerar as RPCs e oferecem ao usuário um ambiente mais amigável.

Em ambos os níveis se realizam execuções remotas com seleção automática da máquina hospedeira e garante-se a transparência destas operações para os usuários do sistema. A escolha das máquinas de destino das migrações baseia-se no nível de ociosidade de suas CPUs.

O sistema visa facilitar o desenvolvimento de aplicações paralelas com baixo custo de implementação, aumentar a velocidade de processamento global de tais aplicações, permitindo o aproveitamento dos recursos das estações ociosas.

*Departamento de Ciência da Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

1 Introdução

Algumas aplicações importantes em engenharia e ciência exigem quantidades de tempo tão consideráveis que são impraticáveis de serem implementadas nos computadores disponíveis. Tais problemas exigem atualmente o uso de supercomputadores ou de arquiteturas concorrentes.

Para que seja possível utilizar, de forma simultânea e efetiva, mais de um processador, realizam-se atualmente pesquisas sobre paralelismo em diferentes áreas:

- Algoritmos e Aplicações,
- Linguagens e Compiladores,
- Arquiteturas, e
- Ambientes de programação.

Este trabalho situa-se nesta última área, que trata do desenvolvimento de ambientes de programação eficientes. Em particular, interessa-nos o desenvolvimento de ferramentas que facilitam a distribuição de módulos para execução em diferentes processadores interligados [BDG+91] [DD92] [Che88].

Um conjunto de estações de trabalho (*workstations*) de última geração interligadas por uma rede local (LAN) de alto desempenho constitui, em essência, um sistema de processadores paralelos fracamente acoplados que poderia ser utilizado como uma "máquina paralela" para certas operações, com o auxilio de ferramentas adequadas. O Sistema Gerenciador de Processamento Cooperativo (SGPC), objeto deste trabalho, implementa uma de tais ferramentas.

2 Projeto do SGPC

O Sistema (SGPC) desenvolvido provê um ambiente que permite ao usuário, de forma relativamente simples, a exploração da capacidade

de paralelismo inerente à arquitetura de rede, para um conjunto de aplicações específicas mas importantes, que satisfazem a determinados prerequisitos.

O SGPC foi desenvolvido sobre uma rede de estações de trabalho *(workstations)* da **Sun**[1] interligadas com uma rede **Ethernet**, e com o Sistema Operacional **SunOS**[2] como plataforma de programação [Sun90]. A programação foi realizada nas linguagens C e RPC[3] [BN84, Sun90]. Esta última foi utilizada na descrição do protocolo de serviço RPC aceita pelo compilador de protocolos *rpcgen*.

A combinação *hardware-software* da rede *Sun* provê uma plataforma flexível para a integração das diferentes máquinas e de novos *softwares*. O Sistema Operacional oferece um conjunto extensível de protocolos para o intercâmbio de dados. Para a instalação e uso do SGPC devem estar disponíveis os serviços de RPC *(Remote Procedure Call)*, XDR *(eXternal Data Reference)*, portmapper, NIS *(Network Information Service)* e NFS *(Network File System)*.

Foi desenvolvido um sistema para distribuir módulos entre estações de trabalho, possibilitando a utilização do tempo ocioso destas. Com a utilização desta ferramenta pode-se conseguir um *"processamento cooperativo"* na rede: não só a utilização compartilhada de periféricos, mas também, da capacidade de processamento das estações.

Para conseguir explorar a concorrência primeiro é necessário verificar se o problema tem paralelismo, e depois dividir a tarefa em processos paralelos, tomando decisões quanto à *granularidade*.

Segmentar um problema em tarefas possíveis de serem executadas concorrentemente é altamente dependente da aplicação. Isto faz com que seja muito difícil, ou impossível, automatizá-lo completamente de uma maneira razoável e eficiente. O SGPC pressupõe que o usuário tem uma idéia clara do paralelismo em seu trabalho e dos algoritmos que os implementam. O SGPC também pressupõe que os módulos possíveis de

[1] Sun Microsystems, Inc.
[2] Sun Operating System
[3] Extensão da linguagem XDR muito similar a C

executar em paralelo residem em arquivos executáveis.

Os usuários podem iniciar módulos independentes que serão migrados, de maneira dinâmica, para serem executados em algumas das máquinas no conjunto disponível. Com a utilização do SGPC passa-se de um ambiente de estações de trabalho distribuídas a um pseudo ambiente de multiprocessadores [KDLS86], fazendo uso efetivo da capacidade de processamento distribuída.

O fato dos módulos serem executados local ou remotamente, bem como a existência de migrações são transparentes para as aplicações. O SGPC pode decidir que a "melhor" estação disponível para executar um módulo seja a própria estação onde foi feito o pedido.

2.1 Módulos do Sistema e sua Interação

Seguindo a recomendação sugerida pelo *SunOS* para implementar serviços de rede, o SGPC baseia-se no conceito de Servidor: um Servidor de Processamento (ServProc) foi implementado como um *"daemon"*[4]; uma cópia deste deve existir em cada estação participando do pseudo ambiente de multiprocessadores. Cada ServProc mantem-se coletando informações e pronto para servir como *Servidor de Processamento*.

Os Servidores nas diversas estações operam concorrentemente, trabalhando para conseguir uma distribuição que equilibre o uso das estações de trabalho, visando minimizar o tempo total de execução. O funcionamento integrado de vários Servidores de Processamento (ServProcs) constitui o Sistema Gerenciador de Processamento Cooperativo.

Os serviços prestados pelo ServProc podem ser usufruídos pelas aplicações usuárias em dois níveis diferentes:

- Sistema Gerenciador de Processamento Cooperativo (SGPC);
- Sistema Gerenciador de Processamento Cooperativo Estendido-(SGPC Estendido);

[4] Processo servidor que fica residente e trabalha em estreita vinculação com o *kernel*

No primeiro caso, a aplicação faz uso do SGPC interagindo diretamente com o Servidor de Processamento Local[5] segundo as convenções do Modelo Cliente/Servidor.

No segundo caso, SGPC Estendido, o SGPC foi acrescentado de uma interface que poupa ao usuário o conhecimento das convenções do Modelo Cliente/Servidor, permitindo que interaja com rotinas simples em uma biblioteca, e com um *front end C*. Esta biblioteca, denominada InterfPC deverá ser incluida, através do ligador, em toda aplicação que use o SGPC Estendido.

Todas as operações disponíveis para a aplicação através da *InterfPC* são mapeadas em serviços do protocolo do ServProc. Neste protocolo estão incluídos além dos serviços propriamente destinados às aplicações usuárias, serviços para a interação entre ServProcs.

Cada ServProc atende a pedidos locais, gerados a partir de aplicações clientes na própria estação, e a pedidos remotos, provenientes de outros ServProcs executando em outras estações.

2.2 Organização e Fluxo das Informações

O SGPC encarrega-se da arbitragem e da alocação de recursos a módulos, e para tanto:

- mantém informações atualizadas sobre as estações na rede;
- mantém informações atualizadas sobre os módulos em execução;
- determina e aloca as estações hospedeiras;
- re-aloca estações para os módulos.

Os Servidores armazenam os dados das estações e dos módulos em execução em tabelas. A manutenção destas tabelas exige a transferência

[5] *ServProc* residente na Estação Local

e distribuição de informações entre as estações participando da cooperação. Nesta primeira versão, optou-se por simplicidade em coletar e enviar sob demanda estas informações.

Existem serviços para dar às aplicações usuárias do SGPC acesso às informações das estações e dos módulos, permitendo a usuários sofisticados adotar critérios próprios para exportações de módulos.

2.3 Controle dos Módulos Submetidos a Migração

A *migração* de módulos acontece entre duas estações em cooperação através dos ServProcs. Diz-se que a "estação origem" do módulo "*exporta*" o módulo. Analogamente, diz-se que a "estação hospedeira" ou "estação destino" do módulo o "*importa*". Como caso particular a estação origem e a estação destino podem coincidir, mas o usuário não precisa ter conhecimento deste fato.

Os pedidos feitos pelas aplicações, segundo a sua natureza, podem satisfazer-se Local ou Remotamente. O percurso das transações passa pela aplicação, pelo ServProc Local e possivelmente por um ServProc Remoto.

Durante a ocorrência destas transações o módulo passa por diferentes estágios: **não_migrado**, **migrado**, **em_progresso**, **concluído** e **erro**.

O tempo de estadia de um módulo no SGPC está marcado por alguns momentos importantes. Seguem-se definições destes tempos:

- **Preâmbulo de Execução**: Tempo entre o reconhecimento do pedido de execução, e o começo desta na Estação Destino. Estados: **não_migrado** e **migrado**.
- **Tempo de Execução**: Período que vai do início do módulo na Estação Destino até o fim de sua execução. Estado: **em_progresso**.
- **Tempo de Vida Estendida**: Inicia-se após a finalização da execução do módulo e dura até que desapareça do controle do SGPC. Estado final: **concluído** ou **erro**. O fim do Tempo de Vida Estendida coincide com o fim do Tempo Total de Vida.

- **Tempo de Vida Ativa**: Inclui o Preâmbulo e o Tempo de Execução. Estados: **não_migrado, migrado** e **em_progresso**;
- **Tempo Total de Vida**: Inicia-se com o recebimento do pedido de execução e se estende até a finalização de seu Tempo de Vida Estendida.

Cada ServProc origem de uma migração atribui um Identificador ao módulo. Este identificador é único para cada módulo no SGPC e válido durante todo seu Tempo Total de Vida.

Enquanto o módulo se encontra no Preâmbulo de execução, apenas o servidor local sabe da sua existência. Durante o Tempo de Execução do módulo, ambos os ServProcs sabem da sua existência, mas é o ServProc Destino que tem controle direto sobre a referida execução; o ServProc Origem apenas é informado de seu estado. Uma vez terminada a execução, o ServProc Origem é informado deste fato, e as informações relativas ao módulo são apagadas do ServProc Destino. No ServProc Origem as informações do módulo sobrevivem ao término da sua execução, daí o Tempo de Vida Estendida.

3 Servidor de Processamento

Um Servidor ou Processo Servidor é um processo que implementa serviços. Um Servidor trabalha em estreita vinculação com o *kernel* do Sistema Operacional, mas seu código não está contido nele.

A criação do ServProc foi auxiliada pelo compilador de RPC *rpcgen*. Este aceita como entrada a definição de uma interface do programa remoto escrita na linguagem RPC, e gera programas na linguagem fonte C.

A saída do *rpcgen* inclui *stubs* do cliente e do servidor do Serviço RPC [BN84]. Estes *stubs* responsabilizam-se pelo *empacotamento* e *desempacotamento* dos parâmetros e resultados, pelo envio da requisição, e pela chamada local do procedimento desejado. As rotinas XDR (opcionais) são úteis no *empacotamento* e *desempacotamento* de parâmetros e resultados em um formato independente da arquitetura dos computadores

envolvidos.

Na geração do código para o SGPC optou-se pelo protocolo de transporte TCP (*Transmission Control Protocol*) devido a sua confiabilidade. Este inclui políticas de retransmissão e *timeout*.

3.1 Serviços do ServProc

Os procedimentos remotos do ServProc podem ser divididos em dois grandes grupos, segundo o tipo do cliente para o que foram preparados:

- serviços para as aplicações Clientes;
- serviços de intercomunicação entre ServProcs.

Através destes últimos materializa-se o intercâmbio de informações de configuração, de informações de estado, e o que é mais importante, a migração dos módulos.

Os serviços no ServProc estão incluídos nos seguintes grupos:

- identidade dos ServProcs:
  - **int ENVIAIDSERVPROC(stamp)**: recebe a Identidade do ServProc cliente;
  - **stamp PEDEIDSERVPROC(void)**: pedido explícito da Identidade do ServProc corrente;
- informações sobre as estações:
  - **ids_host LISTAHOSTS(void)**: retorna uma listagem de todas as estações na rede (usado por aplicações);
  - **inf_host PEDEINFHOST(cadeia)**: oferece características e estado das estações (utilizado tanto por aplicações quanto por outros ServProcs). As informações incluem a arquitetura de aplicação (descrita em 3.3), se a estação tem ou não ServProc residente, a porcentagem de tempo ocioso da CPU, a quantidade de módulos importados, a quantidade de usuários *logados* e a quantidade de processos em execução;

- execução de módulos:
  - **id_mod EXECMOD(mod)**: pedido de execução de um módulo com escolha automática da estação de destino (usado por aplicações);
  - **id_mod EXECMODHOST(mod_host)**: pedido de execução de um módulo com a indicação da estação de destino desejada (usado por aplicações);
  - **id_mod_migrado EXECREMMOD(inf_mod)**: pedido de execução remota de um módulo sendo exportado para esta estação (usado por outros ServProcs). É chamado no corpo da implementação dos dois anteriores;
- informações sobre os módulos:
  - **ids_mods LISTAMODS(void)**: retorna quantos e quais são os módulos migrados através deste ServProc (usado por aplicações);
  - **inf_mod PEDEINFMOD(int)**: oferece o estado do módulo com o Identificador dado como parâmetro;
  - **int ENVIAINFMOD(inf_mod)**: informa à estação de origem de um módulo sobre seu estado na estação de destino (usado por outros ServProcs);
- informações sobre condições errôneas:
  - **inf_erro INFERRO(int)**: dado um código de erro retorna uma mensagem explicativa.

Em todos os casos, as informações retornadas pelos serviços vem acompanhadas da Identidade (*stamp*) do ServProc que prestou o serviço. É responsabilidade do Cliente verificar que as identidades original e corrente coincidam. Nos casos em que os ServProcs fazem o papel de clientes e com o uso do SGPC Estendido, é garantida esta verificação.

Seria desejável que também fossem oferecidos mecanismos de coordenação e interação entre as tarefas. O SGPC, nesta sua primeira versão, não oferece mecanismos para o intercâmbio de informações entre os módulos em execução em diferentes máquinas.

A execução de um módulo começa com a chamada por parte da aplicação de um serviço de execução (**EXECMOD()** ou **EXECMODHOST()**). Com o recebimento destes pedidos os módulos começam seu Ciclo de Vida pelo estado **não_migrado**.

Em ambos os procedimentos de pedido de execução tem que ser dado como entrada o nome do arquivo com o programa executável, o caminho até seu diretório, os argumentos a serem passados ao módulo ao iniciar sua execução, as variáveis de *environment*, se o Servidor Local deve provocar uma coleta de dados novos através de todo o SGPC antes de selecionar a estação de destino, e se o SGPC pode ou não executar novamente o módulo em caso de quedas que interrompam sua execução. Os dois serviços para pedido de execução devolvem o Identificador do Módulo para o SGPC.

O pedido do serviço **EXECREMMOD()** faz com que o módulo transite para o estado **migrado**, ainda no Preâmbulo de sua execução.

Para conseguir a execução do módulo o ServProc faz uso das primitivas *fork()* e *execve()*. O identificador do processo em execução, atribuído pelo Sistema Operacional, constitui o identificador do módulo no destino. O êxito no lançamento da execução do módulo marca o início de seu Tempo de Execução, e como conseqüência o módulo passa ao estado **em_progresso**.

As informações do módulo em execução retornadas por **PEDEINFMOD()** e enviadas por **ENVIAINFMOD()** incluem a estação de destino, os identificadores do módulo no SGPC e no destino, os horários da migração, do reconhecimento do término, do inicio da execução, e do término da execução, assim como o tempo utilizado na execução e o código de saída do módulo.

É através do serviço **ENVIAINFMOD()** que o servidor de origem fica sabendo quando termina a execução de um módulo que exportou, e

o valor de retorno deste serviço é utilizado para conferir se o envio das informações do estado do módulo chegaram a seu destino.

As informações de um módulo são retidas na sua Estação de Destino até que a estação de origem do módulo seja informada de seu término. Entretanto, as informações dos módulos são retidas na sua Estação de Origem ainda depois de terminada sua execução. Um módulo termina seu Tempo Total de Vida no sistema apenas quando a aplicação que provocou sua execução toma conhecimento de sua finalização. Neste caso dá-se por terminado o Tempo de Vida Estendido do módulo.

3.2 Seleção da Estação Hospedeira

A escolha automática da estação de destino para uma migração usa a porcentagem de tempo ocioso (*idle*) da CPU medida desde a última consulta á estação e as migrações anteriores realizadas por este servidor, tentando que não se repitam as seleções.

3.3 Características do Ambiente de Execução

- **Arquitetura de Aplicação**: cada estação de trabalho e cada código objeto gerado por um tradutor tem uma Arquitetura de Aplicação definida. Para um programa ser executado sobre uma estação suas Arquiteturas de Aplicação tem-se que corresponder. O SGPC certifica-se desta correspondência.

- **Interação**: os usuários trabalhando nas estações de destino não devem ser perturbados em seu trabalho por causa dos módulos importados. O módulo migrado não pode usar recursos interativos da Estação de Destino. A execução de uma entrada interativa em um módulo provocaria seu bloqueio. Escritas tanto para a saída padrão (*stdout*) quanto para a saída de erros padrão (*stderr*), são capturadas e redirigidas para arquivos;

- **Variáveis de Ambiente**: a aplicação pode fornecer com o pedido de execução o *environment* que deseja. No SGPC Estendido se o

usuário não oferece este explicitamente o sistema reproduz automaticamente as variáveis do ambiente do programa solicitante;

- **Re-Execução**: em condições excepcionais que impeçam o término normal o SGPC pode executar novamente o módulo.

3.4 Quedas (*Crashs*)

A Identidade de um ServProc é criada na sua instalação a partir do horário do sistema local e do nome lógico da estação. No momento em que se estabelece o primeiro contacto entre dois ServProcs, ambos enviam seus respectivos Identificadores. Em qualquer tentativa de contacto posterior realiza-se uma checagem de sua identidade. No caso de não ser satisfatória a verificação, o ServProc que fez o pedido assume que o Servidor daquela estação caiu e toma as seguintes providências:

Se existem módulos importados no ServProc corrente que foram exportados pelo ServProc considerado em queda, a execução local destes módulos é cancelada.

Se o ServProc corrente exportou algum módulo para a estação em que aconteceu a queda, tenta-se recuperar a situação recomeçando a execução do módulo.

4 Interface de Alto Nível

Os procedimentos da biblioteca que implementa o módulo InterfPC podem ser divididos nos seguintes grupos:

- Ligação ao Servidor de Processamento Cooperativo;
- Operações de Informações sobre as Estações;
- Operações de Execução de Módulos;
- Operações de Informações sobre os Módulos;
- Diagnóstico de erros.

Nas rotinas da InterfPC não se oferecem operações sobre a Identidade dos Servidores de Processamento, uma vez que o uso destes ServProcs fica totalmente transparente para as aplicações usuárias deste nível. Entretanto, o pacote de rotinas nesta biblioteca checa a Identidade dos ServProcs ao longo da Sessão de Processamento Cooperativo.

Nas duas operações de pedido de execução a aplicação usuária tem a opção de fornecer ou não uma cadeia com as variáveis de *environment* e seus valores. Colocando a cadeia vazia como *environment* o SGPC Estendido encarrega-se automaticamente de estabelecer um *environment* igual ao que tinha o processo que pede a execução do módulo.

5 Exemplos de Aplicações

O SGPC oferece serviços que são úteis para aplicações que precisem disparar a execução em lote de vários programas, especialmente quando esses programas vão fazer um uso intensivo da CPU.

5.1 *Make Distribuído*

Na elaboração deste exemplo de aplicação, partiu-se do código fonte de um programa *make* convencional, em que todas as regras devem ser cumpridas seqüencialmente. Neste programa, foram introduzidas modificações para que os pedidos de execução dos comandos que constituem as regras fossem encaminhados através do SGPC e executadas concorrentemente. Este *make* modificado foi utilizado na compilação de conjuntos de arquivos fonte em C, fazendo com que as compilações fossem realizadas de maneira paralela. Cabe ressaltar que o *make distribuído* assim construido foi um simples recurso para realizar os testes de compilação paralela, e não constitui uma ferramenta para uso prático que exigiria extensões que permitissem especificar que regras podem ser executadas em paralelo e qual a sequencialidade que deve ser observada. Nosso *make distribuído* tenta executar todas as regras em paralelo.

5.1.1 Dados de Tempos de Compilações Distribuídas

Apresentamos aqui os resultados obtidos em compilações de arquivos, contendo programas artificiais em C, que requerem aproximadamente 28 segundos cada de uso do processador.

O arquivo fonte tem 4900 linhas de código e foram criadas seis copias do mesmo. É importante assinalar que no sistema utilizado todos os arquivos estão localizados remotamente em um servidor de arquivos.

Foram realizadas compilações de conjuntos de um, três e seis destes arquivos, e variantes destas compilações com o compilador C diretamente, através do *make* original, através do *make distribuído* com o ServProc em uma única estação, e através do *make distribuído* com o ServProc em três e seis estações. As medições de tempo (tempo real decorrido) foram todas realizadas com a ajuda do comando /usr/bin/time. Para cada caso, o experimento foi repetido cinco ou seis vezes e anotados os valores mínimo, máximo e médio. A diferença entre os tempos mínimo e máximo, pode chegar, nos casos em que realmente ocorre exportação, a 54 %, uma vez que os experimentos foram realizados durante a operação normal do sistema, tipicamente por dezenas de usuários.

5.1.2 Compilação de três arquivos de 28 segundos cada

As compilações com medições de tempo foram realizadas nas seguintes variantes:

- **A:** Cada arquivo independente com chamada direta ao compilador;
- **B:** Os três arquivos em uma chamada ao *make* original;
- **C:** Os três arquivos em uma chamada ao *make distribuído* com o ServProc instalado em uma única estação (todas as compilações foram locais);
- **D:** Os três arquivos em uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para três estações diferentes).

Os tempos reais das compilações dos três arquivos, nas diferentes variantes, são mostrados na tabela da Figura 1.

	Mínimo	Máximo	Médio
A	42.40	48.00	44.33
B	82.70	87.90	84.87
C	112.90	116.20	113.90
D	47.00	67.80	56.78

Figura 1: Tempos Reais da Compilação de Três Arquivos

O caso de todas as compilações locais através do *make distribuído* é, como seria de esperar, o de pior desempenho. O tempo aumenta consideravelmente com relação ao do *make* convencional, devido ao *overhead* adicional, natural neste caso, do SGPC. Note-se entretanto que, mesmo neste caso, o tempo total é inferior ao triplo do tempo da compilação isolada.

O resultado em D do uso do *make distribuído* com migrações reais das compilações evidencia melhora no desempenho: embora o tempo de compilação de um módulo migrado seja maior do que se feito localmente, o tempo de compilação do conjunto é cerca de 33 % menor que o tempo necessário para a compilação seqüencial na mesma máquina de todos os módulos.

5.1.3 Compilação de seis arquivos de 28 segundos cada

As compilações com medições de tempo foram realizadas nas seguintes variantes:

- **A:** Cada arquivo independente com chamada direta ao compilador;
- **B:** Os seis arquivos em uma chamada ao *make* original;
- **C:** Os seis arquivos em uma chamada ao *make distribuído* com o ServProc instalado em uma única estação (todas as compilações foram locais);

- **D:** Os seis arquivos com uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para três estações diferentes);

- **E:** Os seis arquivos com uma chamada ao *make distribuído* com vários ServProcs instalados (as compilações migraram para seis estações diferentes).

Os tempos reais das compilações dos seis arquivos, nas diferentes variantes, são mostrados na tabela da Figura 2.

	Mínimo	Máximo	Médio
A	42.40	48.00	44.33
B	167.80	170.10	168.63
C	367.80	745.30	555.17
D	90.70	107.50	97.96
E	79.50	173.80	122.50

Figura 2: Tempos Reais da Compilação de um Conjunto de Seis Arquivos

É importante assinalar que estes exemplos de compilações fazem uso relativamente intenso do sistema de arquivos (em maior medida o de seis compilações) o que introduz demoras que não dependem apenas do processamento na CPU. Note-se que cada compilação resulta na escrita de um arquivo objeto em disco de tamanho considerável.

É de supor que, nestes casos de programas com considerável atividade de entrada/saída, outros fatores tornam-se importantes à medida que aumenta o número de execuções paralelas: no sistema utilizado todas as escritas em disco competem por um único servidor de arquivos central e todo o tráfego passa por uma rede *Ethernet* de 10 Mbps.

Ao aumentar o número das estações de destino de três para seis é mantida a melhora em relação às variantes locais. Mas a melhoria não é proporcional à quantidade de estações. Provavelmente devido aos fatores acima identificados e contrariamente ao que inicialmente se poderia

esperar, em média é pior executar os seis módulos em seis estações ao invés de executar os seis módulos em três estações (dois em cada). Uma explicação plausível para este resultado consiste no fato que, para um maior número de estações atuando em paralelo, estações mais ocupadas tem que ser utilizadas e estas passam a constituir o gargalo do sistema, de vez que o teste só é finalizado após a última estação utilizada concluir seu módulo.

5.2 Aplicações com Uso Intensivo do Processador

Para a realização de testes que façam uso intensivo do processador, foi selecionado o problema de calcular números primos: implementou-se um programa básico extremamente simples e utilizando um algoritmo primitivo, que calcula os números primos em um intervalo. Este programa recebe como argumentos os valores mínimo e máximo dos extremos do intervalo.

Foi desenvolvido um programa principal que, para calcular os números primos em um intervalo, faz uma divisão deste em três subintervalos e chama três instâncias do programa básico. Este programa principal faz uso do SGPC fazendo com que o cálculo dos números primos em cada intervalo constitua um módulo independente possível de ser migrado para sua execução.

Na Figura 3 são apresentados os dados de tempo das variantes, utilizando diretamente o programa básico para o intervalo maior, e para cada um dos três intervalos separadamente.

Os tempos de execução reais para o cálculo dos números primos no intervalo 1 até 8375 foram calculados com quatro abordagens distintas, utilizando:

- **B:** O programa básico;
- **C:** O SGPC com uma única estação como destino das migrações (todas as execuções para o cálculo nos subintervalos foram locais);
- **D:** O SGPC com três estações como destino das migrações (a

Intervalos		Tempos (segs)
Mínimo	Máximo	
1	4750	29.6
4750	6800	30.3
6800	8375	30.1
1	8375	90.70

Figura 3: Tempos Reais de Execução do Cálculo dos Números Primos por Intervalos

execução para o cálculo de cada subintervalo migrou para uma estação diferente);

- **E:** foram provocadas migrações de seis módulos para seis estações diferentes como destino. Para isto, foi executado o programa básico seis vezes com duas cópias para cada subintervalo.

Estes tempos são apresentados na tabela da Figura 4.

	Mínimo	Máximo	Médio
B	89.90	90.20	90.70
C	92.80	96.40	95.17
D	33.9	34.0	33.97
E	35.6	43.9	40.00

Figura 4: Tempos Reais de Execução do Cálculo dos Números Primos do Intervalo Maior

Os resultados confirmam novamente o esperado: a execução local de todos os módulos através do SGPC é a pior opção, embora a desvantagem em tempo não seja muito significativa. Neste caso confirma-se também a notável economia de tempo com o aproveitamento da capacidade de migrações do SGPC.

Diferentemente dos testes realizados com aplicações que fazem uso intensivo de operações de entrada e saída com arquivos remotos, a melhora no desempenho nas aplicações *CPU Bound* é aproximadamente proporcional à quantidade de estações de destino.

6 Conclusões

Foi obtida uma ferramenta prática que permite a utilização de uma rede de computadores como um recurso computacional simples. Esta ferramenta provê um ambiente que facilita ao usuário a exploração da capacidade de paralelismo inerente à arquitetura de rede.

As medidas realizadas com um *make distribuído* simplificado e com uma aplicação *CPU Bound* mostraram a viabilidade da utilização prática e as vantagens que o SGPC pode oferecer aos usuários da rede.

As aplicações usuárias que podem tirar maior proveito do SGPC são aquelas com as seguintes características:

- *CPU Bound*;
- paralelizáveis;
- consumidoras de grande quantidade de tempo.

Note-se que existe um considerável conjunto de aplicações importantes que apresenta as características necessárias para se beneficiarem do SGPC. Uma linha de trabalho em perspectiva é ilustrada a seguir.

Na área de teste de *software* a Análise de Mutantes – um dos critérios da técnica de teste baseada em erros – demanda, para sua aplicação efetiva, uma alta capacidade de processamento para a execução de grande número de mutantes gerados. Uma direção de pesquisa e de trabalho conjunto é integrar o SGPC ao ambiente **PROTEUM** em desenvolvimento no ICMSC-USP [Del92].

O sistema apresentado pode ser um ponto de partida no desenvolvimento de uma ferramenta mais elaborada. Alguns aspectos que podem

ser abordados neste sentido são: permitir interação nos módulos migrados, encaminhando-a à estação de origem, desenvolver uma interface interativa para uso do SGPC, acrescentar facilidades de comunicação entre os módulos, melhorar os critérios de seleção da estação hospedeira e realizar rearbitragem dinâmica.

Referências

[BDG+91] Beguelim, A., Dongarra, J., Geist, A., Marchek, R., and Sunderam, V. A User's Guide to PVM Parallel Virtual Machine. Technical report, Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciences Section, July 1991. version 2.3.

[BN84] Birrel, A.D. and Nelson, B. J. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems*, 2(1):39–59, Feb 1984.

[Che88] Cheriton, D.R. The V distributed system. *Communications of the ACM*, 31(3), March 1988.

[DD92] Drumond, R. and Di Cianni, C. OMNI. sistema de Suporte a Aplica,c oes Distribu'idas. In *VI Simposio Brasileiro de Engenharia de Software, Gramado, RS*, Nov 1992. Projeto A_HAND. Departamento de Ciencia da Computa,c ao.

[Del92] Delamaro, M.E. PROTEUM: Um Ambiente de Teste Baseado na An'alise de Mutantes. Master's thesis, ICMSC/USP, 1992. Exame de Qualifica,c ao de Mestrado.

[KDLS86] Kuck, D.J., Davidson, E.S., Lawrie, D.H. , and Sameh, A.H. Parallel Supercomputing Today and the Cedar Approach. *Science*, 231:967–974, Feb 1986.

[Sun90] Sun MicroSystems Inc. *Network Programming Guide. Sun Release 4.1*, 1990.

Relatórios Técnicos – 1992

01/92 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

02/92 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

03/92 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

04/92 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

05/92 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

06/92 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

07/92 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

08/92 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

09/92 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

10/92 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

11/92 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

12/92 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

01/93 **Transforming Statecharts into Reactive Systems,** *A. G. Figueiredo Filho, H. Liesenberg*

02/93 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *N. das C. Mendonça, R. de O. Anido*

03/93 **Matching Algorithms for Bipartite Graphs,** *H. A. Baier Saip, C. L. Lucchesi*

04/93 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *C. M. H. de Figueiredo, J. Meidanis, C. P. de Mello*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

reltec@dcc.unicamp.br