*Instituto Nacional de Pesquisas Espaciais (**INPE**)*
*Laboratório Associado de Computação e Matemática Aplicada (**LAC**)*
IC / CNPq

RELATÓRIO DE ATIVIDADES

Projeto:

ORIENTAÇÃO A OBJETOS E PROGRAMAÇÃO PARALELA APLICADOS A PROBLEMAS DE COMPUTAÇÃO CIENTÍFICA

Bolsista: Flavio Henrique do Nascimento Moreira

Orientador: Dr. Airam Jonatas Preto (6358)

Período: 01/09/1999 – 30/062/000

Junho de 2000

São José dos Campos – SP

AGRADECIMENTOS

Agradeço a meu orientador e co–orientador, Dr. Airam Jonatas Preto e Dr. Stephan Stephany, pelo apoio incentivo e direção durante todo o período de desenvolvimento desse projeto. Agradeço também ao CNPq pela oportunidade dada a mim, de pesquisar e desenvolver esse projeto e travar um contato pequeno, mas também importante, com a pesquisa e desenvolvimento científico.

1. OBJETIVO

Este relatório descreve as atividades realizadas durante o período da Bolsa PIBIC/CNPq do projeto intitulado "Orientação a Objetos e Programação Paralela Aplicados a Problemas de Computação Científica" desenvolvido por Flavio Henrique do Nascimento Moreira, sob a orientação do Dr. Airam Jonatas Preto, no período entre setembro de 1999 e junho de 2000.

2. PROPOSTA

A proposta inicial do projeto abrange o estudo da aplicação da metodologia orientada a objetos no processamento de alto desempenho, de forma a utilizar objetos distribuídos entre múltiplos processadores, trocando dados e mensagens a fim de efetuar tarefas concorrentes.

Historicamente, as aplicações científicas, principalmente as que estão voltadas ao processamento de alto desempenho, tem sido desenvolvidas em linguagem Fortran. Isso levou ao desenvolvimento de compiladores Fortran extremamente eficientes para a otimização de código, fazendo com que esta linguagem seja considerada uma das mais convenientes para essas aplicações. Dentre as versões de compiladores Fortran disponíveis encontram–se as que se baseiam na especificação HPF (High Performance Fortran). Esta especificação cria diversas diretivas que abstraem a paralelização do programa.

O surgimento da metodologia orientada a objetos (OO) tornou possível o desenvolvimento de programas com facilidade para o tratamento de entidades matemáticas e científicas num alto nível de abstração, que, juntamente com as melhorias na manutenção e reutilização de componentes de programas, tem sido uma motivação crescente para a utilização da programação OO na Computação Científica.

Assim existem várias características do paradigma OO que trazem benefícios para essa abordagem de projeto:

- –Hierarquia de classes: Organiza as classes para reutilização; classes podem herdar características de seus ancestrais;
- –Polimorfismo: Possibilita ao objeto comportar–se, como previsto, diante de tipos de dados diferentes;
- –Expressão Direta da Realidade: Objetos são metáforas naturais tanto para objetos reais como para entidades abstratas;
- –Maleabilidade: o uso de objetos facilita a evolução dos programas através da evolução dos objetos, que encapsulam dados e funções localmente permitindo a modificação com facilidade;
- –Extensibilidade: Usando hierarquia e herança, os objetos podem ser definidos com modificações incrementais;
- –Abstração: Usando polimorfismo, similaridades entre objetos podem ser expressadas no programa, permitindo escrever código em termos de similaridades sem se importar com as diferenças.

Deve–se considerar ainda o fato do paradigma OO ser baseado no conceito de objetos que interagem enviando e recebendo mensagens. Esta característica permite que um conjunto de objetos seja mapeado para um grupo de processadores, para que seja explorado o paralelismo nas suas computações.

3. IMPLEMENTAÇÃO

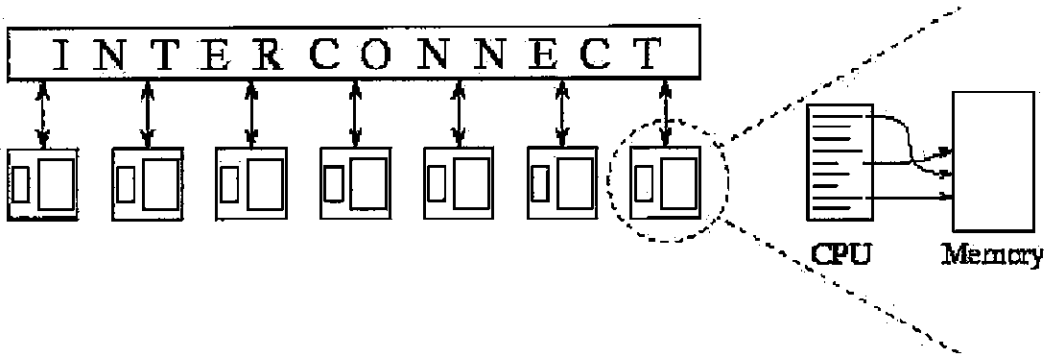
i. Plataforma:

Para a implementação desse projeto deve–se considerar, antes de tudo, a plataforma de hardware a ser utilizada. Os computadores paralelos mais comuns se classificam, nas seguintes categorias:

–Os computadores de Memória Compartilhada, são aqueles nos quais uma única memória é compartilhada e acessível entre todos os processadores. Toda a memória tem um espaço de endereçamento único, e portanto pode ser acessada por operações simples de LOAD/STORE. Essa arquitetura pode ser constituída por um sistema multiprocessado, ou até mesmo por um sistema constituído por múltiplos computadores conectados que possuam uma camada de software que implemente a memória compartilhada.

–Os computadores de Memória Distribuída, são aqueles nos quais cada nó de processamento possui sua própria memória com seu espaço de endereçamento individual. Portanto cada nó só tem acesso a memória distribuída por meio de operações SEND/RECEIVE, através do uso de uma biblioteca ou interface de comunicação por mensagens.

Nossa plataforma de estudo se baseia numa abstração do 2o caso, pois utilizamos bibliotecas de comunicação por mensagens, em uma estrutura como da figura seguite:



Esta será baseada em estações Unix que podem ser configuradas como um conjunto de processamento, que permite estabilidade, disponibilidade e flexibilidade. Sendo implementada com uma estrutura de computação paralela construída a partir de estações Intel x86 rodando Linux com kernel 2.2, que suporta SMP (Multiprocessamento Simétrico), utilizando uma biblioteca para comunicação por mensagens, MPI (message passing interface).

A biblioteca MPI é composta por funções atômicas para tarefas genéricas de troca de mensagens/dados, que permitem a implementação de programas paralelos voltados para uma ampla gama de problemas. Em nosso caso especifico, será usada uma extensão da biblioteca MPI denominada OOPMI, que implementa as funções de comunicação por mensagem usando a abstração de Objetos.

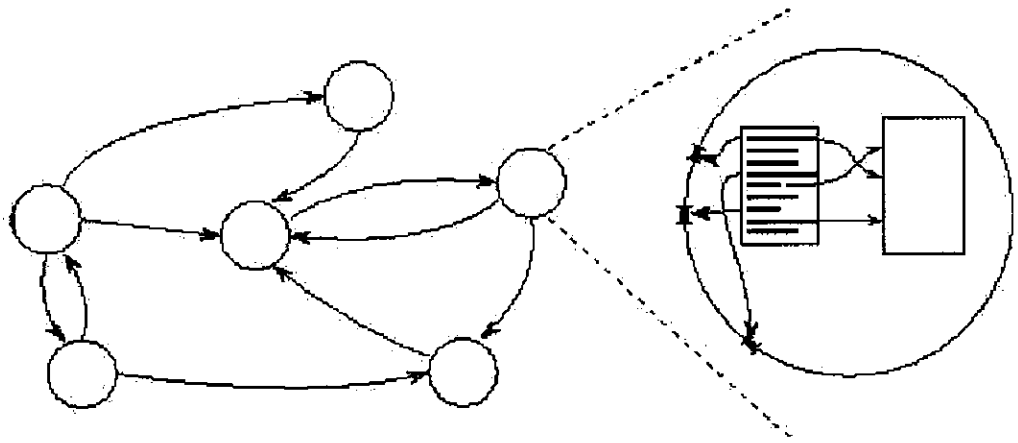
Especificamente, o sistema utilizado nas estações Linux foi configurado para fazer uso da distribuição do MPI conhecida como MPICH e a implementação especifica para OO, conhecida como OOMPI.

No total, foi possível disponibilizar nessa plataforma o uso de até 6 processadores, quatro utilizados de uma máquina quadri–processada utilizando SMP e OOMPI e dois de maquinas mono–processadas utilizando OOMPI.

ii. Programa:

Paralelização:

No que tange a sua forma de implementação, um programa paralelo pode ser divido em tarefas que processam dados localmente, sem a necessidade estabelecer conexão com outros processos – que são executados de forma concorrente – e em canais de comunicação que cada tarefa ou processo abra para um ou mais processos e que, portanto, sirvam para a troca de dados ou informações. Como pode ser visto na figura seguinte:

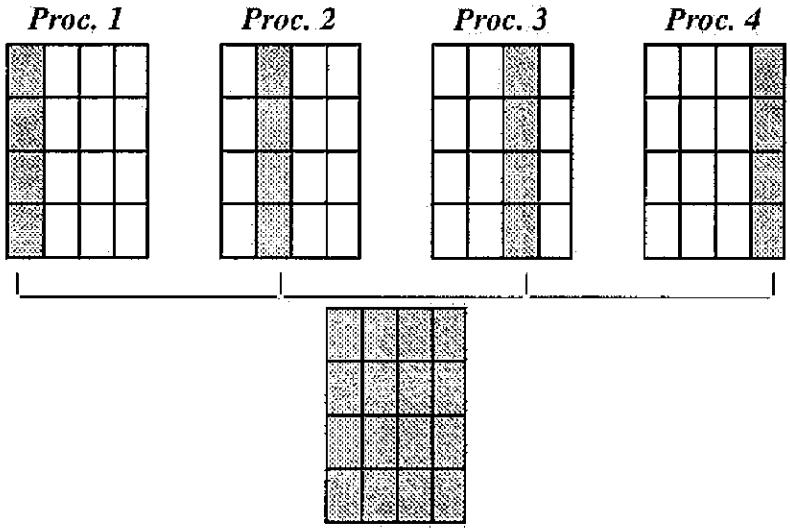


Dentro desses princípios existem duas abordagens para desenvolvimento de programas paralelos:

–Paralelismo de Tarefas: nessa abordagem o programa é divido em várias tarefas individuais que podem ser executadas de forma concorrente e essas tarefas são distribuídas entre vários processos. Em alguns casos, porém, existe a necessidade de criar um sistema de sincronização entre os vários processos para que tarefas que são interdependentes possam ser executadas paralelamente;

–Paralelismo de Dados: essa abordagem é aplicável quando um grande conjunto de dados pode ser dividido entre os vários processadores de um computador paralelo, para serem processados por rotinas distribuídas em cada processador, as quais podem ser iguais sobre todos os processadores ou diferentes para cada um deles.

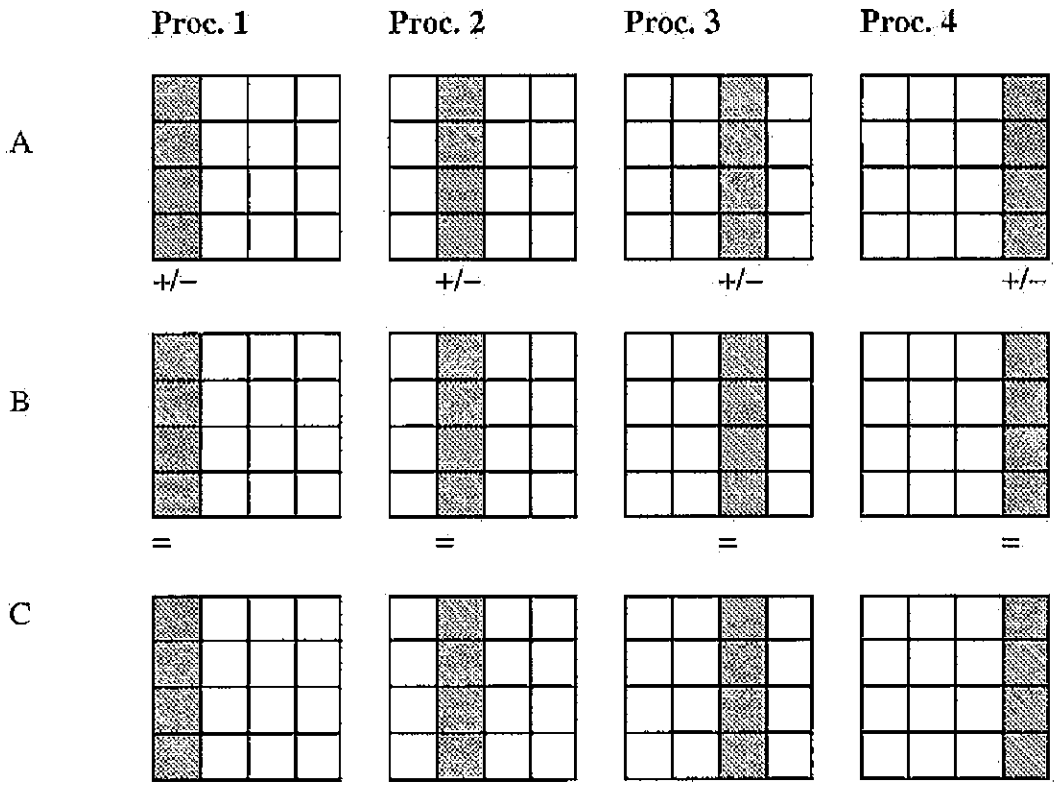
Nesse caso, a comunicação entre processos será executada, como já citado anteriormente, pela biblioteca de comunicacao MPI. A respeito da abordagem de paralelismo utilizada, deve-se considerar que esse projeto resultou na implementação de uma classe Array de duas dimensões, que encapsula as operações matemáticas efetuadas pelas matrizes (como soma, subtração, multiplicação por escalar e por outra matriz), assim, de forma a conseguir operações paralelas transparentes durante a programação, utiliza-se o Paralelismo de Dados, a fim de distribuir entre os vários processadores da plataforma os elementos do objeto Matriz. Como pode ser visto na figura a seguir:



Para cada uma das operações matemáticas possíveis para uma matriz, analizou-se as otimizações aplicáveis e suas respectivas distribuições de dados necessárias, para que se conseguisse uma melhora de desempenho. Ou seja, se uma soma de matrizes for efetuada em objetos que tenham distribuições ótimas, esta será efetuada de forma mais eficiênte do que no caso de distribuições de dados caóticas, e da mesma forma para todas as outras operações. As otimizações foram feitas desta forma:

Soma e Subtração:

A operação de soma ou subtração entre matrizes ocorre elemento a elemento, portanto, a única otimização necessária, quanto a distribuição das matrizes, é que elas tenham distribuições de dados exatamante iguais, assim os elementos que tem que ser operados estarão sempre no mesmo processador, evitanto comunicação e, portanto, perdas. E acima de tudo, os cálculos podem ser executados concorrentemente, cada processador sobre seu conjunto de dados. Assim utilizando-se, por exemplo, uma distribuição de elementos similar a mostrada na figura anterior, teremos um processo de cálculo como mostrado na figura a seguir:



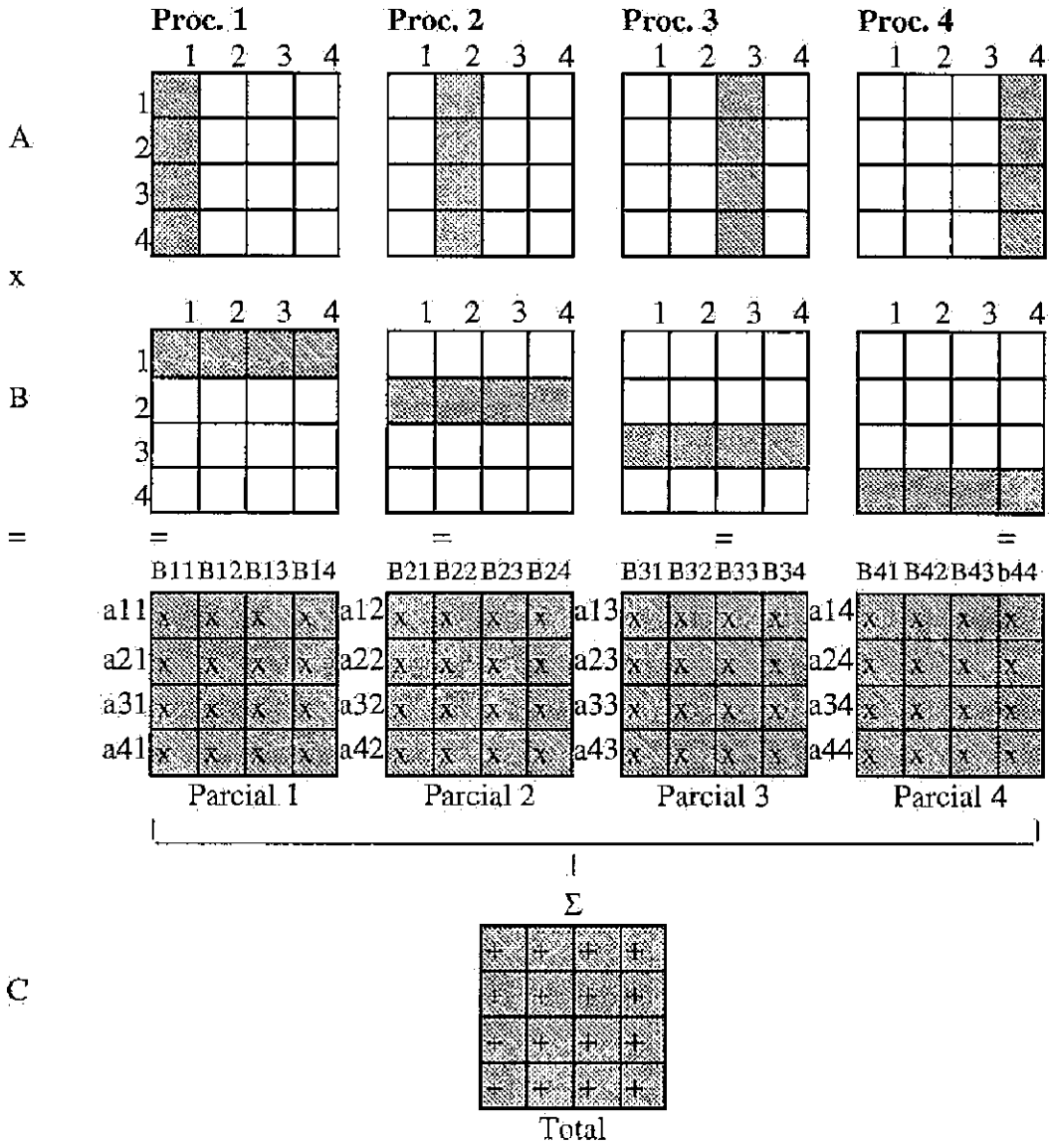
Assim temos: Cij=Aij+Bij. Como cada processador possui seu conjunto de dados temos uma soma eficiente. Caso a distribuição dos elementos do Array não fosse ótima, teríamos que passar cada element, por interface de comunição, entre os processos para que estes pudessem ser somados o que causaria uma grande perda de desempenho

Multiplicação Matriz X Matrix:

A multiplicação Matriz X Matrix, envolve uma complexidade maior, pois o algoritmo para multiplicação desse tipo exige interacao muito grande entre os elementos de ambas as Matrizes. Se implementada da maneira tradicional essa operação sofreria com um grande volume de comunicação entre os processadores, o que prejudicaria irremediavelmente o desempenho do programa, e portanto todas as vantagens da implementação paralela estariam perdidas.

Assim para a implementação dessa operação utilizou uma alternativa similar a multiplicação blocada, ou seja, a partir de distribuições transpostas dos elementos das matrizes, faz–se multiplicações locais em cada processador conseguindo produtos parciais que depois passam por um processo de soma.

Esse processo, e a distribuição como seria necessária, podem ser melhor visualizado na figura a seguir:



Assim tempos: Pj = Aij x Bji [j=RANK, i=1..n]
Total = Σ Pj [j=1..m]

Logo, obtemos uma forma de multiplicação de matrizes dividida entre todos os processadores com só um passo de comunicação que ocorre no momento da totalizacao das Parciais.

Esse método só é efetivo se a distribuiçao dos elementos for obedecida fielmente, caso isso não ocorra esse método é ainda mais lento que o tradicional e portanto aquele deve ser usado no lugar deste.

Multiplicação Matriz X Escalar:

A multiplicação de uma matriz por um escalar é uma operação bem mais simples, não importando qual a distribuição dos elementos aplicada sobre os processadores, a operação pde ser otimizada. Basta que o valor escalar pelo qual se deseja multiplicar a matriz seja comunicado a todos os processadores (através de um broadcast) e, então, cada um dos processadores efetua a multiplicação sobre o seu subconjunto de elementos.

Linguagem:

A propósito da linguagemde programação utilizada, o C++ foi escolhido para o desenvolvimento dessa classe, principalmente por ser a única linguagem compatível com o MPI que permite Orientação a Objeto, além das suas seguintes características:

- Disponibilidade: A linguagem é encontrada em um amplo espectro de plataformas;
- Portabilidade: Além de estar disponível em muitas plataformas a grande maioria dos compiladores segue quase completamente uma especificação única (baseada na especificação de B. Stroustrup [5]) , logo o código é facilmente portável para a maioria dos compiladores;
- Eficiência: O C++ foi projetado desde de o início para manter a eficiência compatível com a do C;
- Correção: É uma linguagem que captura praticamente todos os erros em tempo de compilação ou ligação, pois desestimula, algumas práticas inconvenientes que eram fontes de erro no C;
- Generalidade: O C++ é uma linguagem de uso geral que pode ser aplicada a uma ampla gama de problemas;
- Bibliotecas: É possível acessar, direta ou indiretamente, bibliotecas C ou Fortran, portanto pode–se aproveitar uma vasta gama de bibliotecas já testadas.

iii. Atividades:

Inicialmente os estudos efetuados foram voltados a aprendizagem da linguagem Fortran 77 (em [2]), que é considerada uma linguagem clássica para resolução de problemas de computação científica e apresenta uma grande quantidade de algoritmos já implementados, e que, portanto, servem como uma base para os algoritmos e modelos que serão desenvolvidos durante o projeto. Esse estudo também foi dirigido ao posterior aprofundamento na linguagem objetivando o aprendizado de especificações mais modernas do Fortran.

Um aspecto importante a ser destacado é que a linguagem Fortran já possui bibliotecas para desenvolvimento científico e matemático, bem como, é extremamente otimizada para essa tarefa, logo constitui-se numa referência para parametrização de estudos comparativos entre rotinas de cálculo usando C++. Além dessas características, a linguagem Fortran foi escolhida como base de estudo por possuir compiladores com implementação de processamento paralelo (HPF), o que pode ser usado como referência para testes com rotinas desenvolvidas em C++/MPI (usando a biblioteca OOMPI).

Após o estudo e implementação de algoritmos matemáticos básicos usando Fortran 77, passou-se ao aprendizado do HPF (High Performance Fortran) que estende o Fortran 90 para máquinas paralelas. Logo, pode-se comparar o desempenho de programas compilados tanto para processamento paralelo como para processamento serial, pois a especificação HPF permite que programas possam ser compilados tanto no HPF como no Fortran 90.

Durante o aprendizado do HPF foi necessário o estudo mais detalhado de alguns conceitos relativos as máquinas e a programação paralela, e como visto em [3] temos que, todo o desenvolvimento e estudo da ciência da programação de computadores está baseado no clássico modelo da arquitetura de Von Neumann. Esse modelo dita que as instruções são lidas de uma memória, executadas sequencialmente e seu resultado é também posto na memória. No entanto quando se fala em programação paralela, esse esquema de programação fica comprometido, em dois aspectos:

1. As instruções podem não vir/ir de ou para uma única memória.
2. Por definição a computação paralela determina que várias instruções podem ser executadas simultaneamente.

Como já citado um programa paralelo deve ser divido em tarefas que processam dados localmente, e em canais de comunicação que cada tarefa ou processo abra para um ou mais processos e que portanto sirvam para a troca de dados ou informações.

Na linguagem HPF é utilizado o paradigma de processamento paralelo conhecido como paralelismo de dados, ou seja , os dados são distribuídos entre os diversos processadores e o programa é executado em todos os processadores igualmente, sobre conjuntos de dados distintos. Esse esquema é conhecido também como SPMD (single program multiple data).

Um próximo passo foi o estudo do C++ .Para este estudo foi utilizado o texto de Barton e Nackman [1], que além de apresentar os conceitos básicos do desenvolvimento OOP usando C++ , enfoca principalmente o desenvolvimento de classes matemáticas e científicas e aproveitamento (encapsulamento) de rotinas e bibliotecas matemáticas já existentes em objetos, este último tópico, por não ser o objetivo primário do trabalho, foi superficialmente estudado.

Durante este estudo, foram verificados sobretudo os meios e estruturas utilizadas para o desenvolvimento de entidades de aplicação científica, principalmente classes que abstraem arrays e suas operações. Foram feitos alguns testes práticos com as classes fornecidas em [1] e verificada a flexibilidade e facilidade advinda dessas construções.

Finalmente, foram verificadas aplicações C utilizando biblioteca MPI, visando futuramente a aplicação dessa biblioteca à programação C++. O objetivo principal desse período foi familiarização com as principais diretivas MPI e construção de programas paralelos básicos, como a integração pelo método do trapézio e o método de simpson, utilizando computação distribuída.

Estes testes realizados mostraram a eficiência da biblioteca MPI no ambiente de desenvolvimento descrito, no entanto ainda não foram realizados experimentos com a utilização do MPI (ou biblioteca similar) em ambiente de desenvolvimento C++, pois esse será um dos próximos passos do projeto.

Num estágio posterior as atividades foram voltadas a aplicação do MPI ao desenvolvimento de classes matemáticas em C++. Fez-se o estudo dos meios de aplicação da biblioteca MPI em comunicação e passagem de mensagens entre em grupos de estacões operando em processamento paralelo. Essa abordagem difere da abordagem comumente usada para a paralelização usada em MPI pois se baseia em troca de mensagens entre objetos, e não na paralelização de dados.

Então passou-se ao estudo da biblioteca de classes conhecida como OOMPI, que encapsula as diretivas MPI em classes de objetos. Essa biblioteca provê um nível maior de abstração no desenvolvimento de conjunto de classes que operam distribuidamente, permitindo então um código mais limpo, livre de diretivas C puro.

Finalmente, partimos para a implementacao da classe Array2D que chamou-se PMatrix<T> e possui a seguinte interface:

```
// Classe ARRAY 2D PARALELA

const int P_NONE=     0;
const int P_BLOCK=    1;
const int P_RING=     2;
const int P_COLAPSE= 3;


// interface do objeto classe

template<class T>
   class PMatrix {
```

```
private:
    int _Rows;
    int _Cols;
    T* m;
    int _flag;

    // Dados de Distribuicao
    int RowDistribution;
    int ColDistribution;
    int RowBlockSize;
    int ColBlockSize;

    // Variaveis Comuns a todos os processos
    int _rank;
    int _size;

    // funcao de localizacao de elementos
    int Distrib_Loc(int x, int y, int tempsize);

public:
    // Tipo de dado do Array
    typedef T  P_Type;

    // Construtores
    PMatrix(int nR, int nC, int dR=P_NONE,
            int dC=P_NONE);
    PMatrix(PMatrix<T> &t2, int dR=P_NONE,
            int dC=P_NONE);
    PMatrix():_Rows(0), _Cols(0), m(0), _flag(0),
              RowDistribution(0), ColDistribution(0)
    {
        _rank = OOMPI_COMM_WORLD.Rank();
        _size = OOMPI_COMM_WORLD.Size();
    };

    // Destrutor
    ~PMatrix(void)  { if(m) delete [] m; }

    // Metodos p/ reinicializacao
    int Dim(int nR, int nC);
    int Dim(PMatrix<T> &t2);

    // Metodos p/ leitura dos Paramentros internos
    inline int  Rows()    { return _Rows; }
    inline int  Cols()    { return _Cols; }
    inline int numElts() { return (_Cols*_Rows); }

    // Sobrecarga de Operadores
    PMatrix<T>& operator +(PMatrix<T> &);
    PMatrix<T>& operator -(PMatrix<T> &);
    PMatrix<T>& operator *(T cte);
    PMatrix<T>& operator *(PMatrix<T> &);
    PMatrix<T>& operator =(PMatrix<T> &);
    PMatrix<T>& operator =(T cte);
```

```
    // Metodos p/ Distribuicao de dados
    T   Get (int x, int y);
    int Set (int x, int y, T valor);
    int Distribute(int row, int col);

}; // FIM DA INTERFACE DA CLASSE
```

Após verificar a funcionalidade dessa classe e constatar que esta estava operacional, fez–se alguns levantamentos de desempenho em um número variado de processadores e obteve–se a seguinte planilha de resultados:

SOMA

Dimensão / Procs	64	256	1024	2048
1	0,017842	0,285146	4,575170	59,220400
2	0,016222	0,258820	4,144050	51,983000
4	0,015508	0,247383	3,916740	43,162600
6	0,015985	0,245336	3,842690	40,021100

Dimensão / Procs	64	256	1024	2048
1	100,00%	100,00%	100,00%	100,00%
2	90,92%	90,77%	90,58%	87,78%
4	86,92%	86,76%	85,61%	72,88%
6	89,59%	86,04%	83,99%	67,58%

SUBTRA

Dimensão / Procs	64	256	1024	2048
1	0,009223	0,169625	2,562710	10,286700
2	0,009182	0,171238	2,888210	83,424600
4	0,009149	0,204858	3,448420	370,139000
6				

Dimensão / Procs	64	256	1024	2048
1	100,00%	100,00%	100,00%	100,00%
2	99,56%	100,95%	112,70%	810,99%
4	99,20%	120,77%	134,56%	3598,23%
6	0,00%	0,00%	0,00%	0,00%

MULTIPLICACAO

Dimensão Procs	64	256	1024	2048
1	0,084453	14,877900	2,562710	1369,900000
2	0,086482	13,897500	2,888210	1369,150000
4	0,095217	14,267500	3,448420	1368,950000
6				

Dimensao Procs	64	256	1024	2048
1	100,00%	100,00%	100,00%	100,00%
2	102,40%	93,41%	112,70%	99,95%
4	112,75%	95,90%	134,56%	99,93%
6	0,00%	0,00%	0,00%	0,00%

4. CONCLUSÃO

Após o desenvolvimento desse projeto, considero que, grande parte dos conceitos relacionados a processamento de alto desempenho tenham sido absorvidos e aplicados. No entanto a Classe PMatrix<T> desenvolvida durante o projeto para aplicação da biblioteca OOMPI , e o expeimento de objetos rodando de forma distribuida em um conjunto de processadores, ainda carece de algumas otimizações, para as quais não houve tempo habil. Nota-se, pelo levantamento de desempenho dessa classe que, quando aplicadas diferentes operações em números diversos de processadores atuando conjuntamente, somente no caso da soma de matrizes com distribuição ótima dos elementos há um ganho real de desempenho, aliás curiosamente na subtração de matrizes, que utiliza o mesmo algoritmo da soma, a perda de perfromance e grande mesmo para a distribuição ótima. Quanto à multiplicação de matrizes os melhores resultados obtidos resultavam em um empate de desempenho para qualquer número de processadores, e portanto um grande desperdicio de recursos. Logo é clara a necessidade de repensar-se ainda a estrutura dessa classe, talvez substituir a biblioteca OOMPI pela MPI original, pois assim contornariamos algumas prováveis perdas causadas pela implementacao dessa biblioteca; uma outra sugetão seria a alteração do método de acesso dos elementos em cada processador, implementando-o de forma indexada, assim talvez se obtenha pequenos ganhos para a soma e multiplicacao. Finalmente para a subtracao, não há nada de conclusivo que possa ser aplicado para a sua melhoria, a única suposicao plausível é a de que, pelo fato da operação ser uma soma em complemento de dois, essa demandaria mais tempo de processamento que somada a mínima comunicacao necessária para distribuição seria maior que o ganho da paralelização.

São José dos Campos, 30 de junho de 2000

Flavio H. N. Moreira

Flavio H. N. Moreira

Ciente e de acordo:

Airam Jonatas Preto

5. REFERÊNCIAS

1. Barton, J.J.; Nackman, L.R.
   Scientific and Engineering C++.
   Addison–Wesley Publishing Co., USA, 1994.

2. Ellis, T. M. R.
   Fortran 77 Programing.
   Addison–Wesley Publishing Co., USA,

3. Foster, Ian
   Designing and Building Parallel Programs.
   Addison–Wesley Publishing Co., USA,

4. Pacheco, P.
   Parallel Programming with MPI.
   Morgan Kaufmann Publishers, USA, 1996.

5. Stroustrup, B.
   The C++ Programming Language.
   Addison–Wesley Publishing Co., USA, 1991.