Medidas de Desempenho na Máquina Vetorial CRAY Y-MP2E.

G. Wirth[1], P. Navaux[2] e S. Bampi[3].

Abstract - This study deals with benchmark programs and factors that affect the performance of computer systems.

The performance of different computer systems, including the CRAY Y-MP2E, was measured using standard and locally developed benchmark programs.

A number of pitfalls that users have to be aware of, while comparing performance of computer systems, are characterized.

Resumo - Este trabalho trata da técnica de obter medidas de desempenho de máquinas e dos fatores que influenciam o desempenho dos programas executados nestas máquinas.

O desempenho de diferentes máquinas, incluindo o CRAY Y-MP2E foi medido, usando-se programas amplamente difundidos e outros localmeente desenvolvidos.

Um conjunto de fatores com os quais os usuários devem tomar cuidado, ao comparar o desempenho de máquinas, são caracterizados.

1 - INTRODUÇÃO.

Sempre dispensou-se bastante atenção à técnicas de benchmark. Este interesse por medidas de desempenho tem seu principal fundamento na necessidade de comparar-se sistemas diferentes, a fim de definir qual deles apresenta a melhor relação custo/beneficio, no caso de uma eventual compra de equipamentos. As medidas de desempenho também são importantes para comparar-se a adequação de diferentes sistemas/métodos à solução de um problema específico.

2- DESEMPENHO MÁXIMO DE UMA MÁQUINA.

Toda máquina possue um desempenho máximo possível de ser atingido, limitado pelo seu Hardware. Este desempenho máximo pode ser estimado em condições ótimas, como valor teórico, isto é, calculado e não medido. Este valor representa a "velocidade da luz" para uma determinada máquina, ou seja, nenhum programa poderá obter um desempenho acima deste valor.

Por exemplo, para o Cray Y-MP 2E/232 disponível no Centro Nacional de Supercomputação, com dois processadores, de ciclo de clock de 6 ns, sendo que cada processador é capaz de realizar duas operações de ponto flutuante por ciclo de clock (uma soma e uma multiplicação), temos o seguinte desempenho máximo teórico:

$$\frac{2 \text{ F. P. Op.}}{1 \text{ cycle}} * \frac{1 \text{ cycle}}{6 \text{ ns}} * 2 \text{ CPUs} = 666{,}7 \text{ MFLOPS}$$

No Cray Y-MP 2E/232 nenhum programa poderá obter um desempenho superior a 666,7 MFLOPS. Os fatores que determinam o desempenho realmente obtida por programas no Cray é o objeto deste trabalho.

O desempenho de I/O não foi estudado ou comparado neste trabalho, já que os benchmarks utilizados são intensivos em cálculo. O CRAY Y-MP da Centro Nacional de Supercomputação tem um

[1]Estudante de mestrado junto ao CPGCC-UFRGS. Porto Alegre, R.S. E-Mail: wirth@inf.ufrgs.br.
[2]Professor junto ao CPGCC-UFRGS. Porto Alegre, R.S. E-Mail: navaux@inf.ufrgs.br.
[3]Professor junto ao CPGCC-UFRGS e diretor do Centro Nacional de Supercomputação (CESUP). Porto Alegre, R.S. E-Mail: bampi@inf.ufrgs.br.

subsistema de I/O com 2 clusters e 8 canais de I/O. A memória RAM compartilhada pelos 2 processadores está organizada em 64 bancos e 4 seções, num total de 32 MWords (256 Mbytes).

3 - MÉTODOS PARA MEDIDA DE DESEMPENHO.

Podemos dividir os métodos de medida de desempenho em três grandes grupos.

3.1 - Medir o desempenho obtido em um "run" do SW.

Este método é utilizado basicamente quando deseja-se medir o desempenho obtido por um programa não desenvolvido com a finalidade de medir desempenho de máquinas. Para tanto, utiliza-se um segundo programa que "monitora" a execução do primeiro, como por exemplo o utilitário hpm (*hardware performance monitor*) no Cray.

Quando utiliza-se este método para aferir-se desempenho, o valor obtido não refere-se ao desempenho de uma característica especifica da máquina (como por exemplo o desempenho da unidade de ponto flutuante). Na medida do desempenho entram instruções de varios tipos. Entretanto, alguns programas utilizam com maior intensidade uma determinada característica da máquina, sendo assim, o desempenho obtido estará fortemente relacionada a esta caraterística da máquina. Por exemplo, o desempenho obtido por um programa para a resolução de sistemas de matrizes está fortemente ligado ao desempenho da unidade vetorial de ponto flutuante da máquina.

É importante salientar que na medida de desempenho realizada pelo programa que monitora o desempenho real geralmente não se computa o tempo gasto com I/O.

3.2 - Medir o desempenho de laços ou rotinas.

Este método consiste em medir-se o desempenho obtido em um determinado trecho do programa. Para que isto seja possível, devemos possuir uma função para medir intervalos de tempo real. Em algumas máquinas esta funcão é fornecida pelo sistema operacional.

Este método permite que se avalie o desempenho obtido para determinadas operações, como no exemplo abaixo:

```
      TM = CPTIME ()                        (1)
C
C  TIMING TEST
C
    DO 121 II = 1, IT                       (2)
     DO 111 J = 1, M                        (3)
      DO 111 K = 1, N                       (5)
       DO 111 I = 1, L                      (6)
        C(I,K) = C(I,K) + A(I,K) * B(I,K)   (7)
 111  CONTINUE                              (8)

 121  CONTINUE

 TM = CPTIME ()                             (9)
 ER = ABS ((C(19,19) - ANS) / ANS)          (10)
 FP = 2 * IT * L * M * N                    (11)
C
C  PERFORMANCE EM MFLOPS
C
 RT = 1E-6 * FP / TM                        (12)
```

Exemplo 1.

No exemplo acima, desenvolvido para o CRAY, avalia-se o desempenho da unidade de ponto flutuante, da vetorização e paralelização (quando a máquina as oferece).

A função CPTIME() retorna o tempo transcorrido desde a última chamada a esta função. Portanto, o tempo gasto em executar as operações contidas entre (2) e (9) estará armazenado em TM, depois da execução de (9).

Nota-se que no loop DO mais interno, a sentença (7) do exemplo acima, realiza duas operações de ponto flutuante, uma soma e uma multiplicação (que no caso do Cray são realizadas em paralelo). Logo, o número de operações de ponto flutuante realizadas é duas vezes o número de vezes que a sentença (7) é realizada, ou seja, 2*IT*L*M*N.

Assim, o desempenho obtido, em milhões de operações de ponto flutuante por segundo (MFLOPS), na execução do loop (7) é calculado em (12).

Observe-se que caso haja interrupções enquanto as operações contidas entre (2) e (9) estiverem sendo executadas, o desempenho calculado sera menor que o real, porque o tempo gasto realizando operações de outro processo será computado. Todos os Benchmarks rodados tem este problema. Por isso, executou-se o programa várias vezes e observou-se o resultado. Não houve diferença substancial entre os vários "runs".

O método de medir o desempenho de laços ou rotinas é o método utilizado pelos programas escritos especificamente para a medida de desempenho, como por exemplo:

i) O Dhrystone [4]: Benchmark muito popular desenvolvido inicialmente em linguagem ADA, sendo a versão mais popular escrita em C. Foi desenvolvido para medir o desempenho de pequenas máquinas com arquiteturas simples.

O teste gasta significativo tempo com funções do tipo string. As máquinas baseadas em arquiteturas RISC levam uma grande vantagem sobre as CISC porque possuem um grande número de registradores que são utilizados por compiladores otimizados. Um grande problema é sua dependência com a linguagem utilizada, as funções que manipulam string ocupam 10% do tempo de processamento em PASCAL e 16% em ADA [1].

ii) O Whetstone [5], [6]: Primeiro programa escrito com propósitos de benchmark. Sua primeira versão foi escrita em ALGOL. Como o Dhrystone, o Whetstone é um programa simples e pequeno. Sendo assim, qualquer otimização resultará em grande diferença no desempenho final do sistema. Alguns fabricantes retiram a rotina de impressão dos resultados intermediários para melhorar o desempenho [1].

iii) O Linpack [7], [8]: Coleção de subrotinas de álgebra linear utilizadas para caracterizar o desempenho de máquinas operando em ponto flututante. Este benhcmark é bastante sencível ao tamanho da memória cache, da memória principal no que se refere ao tamanho e a organização desta e a otimizações do compilador. O código é bastante vetorizável.

iv) O Livermore [9], [10]: Também chamado de Lawrence Livermore Loops, cosiste de 24 kernels de diferentes áreas da ciência. Os kernels representam casos reais de problemas em ponto flutuante. O programa faz medidas para três diferentes tamanhos de vetor. Como resultado são apresentadas as médias aritmética, hormônica e geométrica, bem como o máximo e mínimo desempenho obtidos.

v) O NAS [3]: O NAS (Numerical Aerodynamic Simulation) Kernel Benchamark Program foi desenvolvido pelo NASA Ames Research Center, consistindo de 7 kernels contendo aproximandamente 1000 linhas de código fonte em linguagem FORTRAN [3]. As rotinas foram retiradas dos projetos desenvolvidos por este órgão, envolvendo projetos de dinâmica dos fluídos, e tratam de problemas como: i) multiplicação de duas matrizes (MXM); ii) FFT complexa de raiz 2 em uma matriz bi-dimensional (CFFT2D); iii) decomposição Cholesky paralela em um conjunto de matrizes (CHOLSKY); iv) solução de matriz tridiagonal em bloco ao longo de uma dimensão de uma matriz de dimensão quatro (BTRIX); v) eliminação Gaussiana em conjuntos de matrizes (GMTRY); vi) aplicação do método Vortex para gerar vertices de acordo com condições de contorno específicas (EMIT) e (vii) inverção simultânea de três matrizes pentadiagonais (VPENTA).

3.3 - Realizar Projeções.

Este método parte do desempenho conhecido em uma determinada máquina, e faz a estimativa do desempenho projetado em outra, através da comparação das características das duas máquinas. Este método supõe que, por exemplo, conhecendo-se o desempenho de um modelo de um determinado sistema, quando quadruplicarmos o tamanho do sistema (número de processadores, memória, etc), o desempenho será quatro vezes maior (projeção linear).

4 - FATORES QUE INFLUENCIAM O DESEMPENHO.

O desempenho de uma aplicação em uma determinada máquina depende de vários fatores, dos quais destacam-se:

4.1 - Compilador:

A eficiência do compilador pode limitar drasticamente o desempenho de uma máquina. Além disto, muitos compiladores permitem três níveis de interação com o programador:

4.1.1 - Compilação Automática:

Não há interação nenhuma entre usuário e o compilador. O compilador escolhe a melhor maneira de compilar o código a ele submetido. Para o usuário, esta é a opção mais confortável, já que ele não necessita conhecer a arquitetura alvo. Mas infelizmente, o código gerado desta maneira geralmente não é muito otimizado, atualmente.

4.1.2 - Compilação Semi-Automática.

Utiliza-se opções de compilação na linha de comando. Através destas opções indica-se ao compilador quais de seus módulos devem se acionados ou não. Como por exemplo no compilador CFT77 Fortran do Cray a opção -Z, que indica quais as fases do compilador devem ser executadas ou não, e a opção -W passa parâmetros para cada uma das fases

Com a utilização destas opções já é possivel obter-se um aplicativo mais otimizado, sem que o usuário tenha que conhecer bem a arquitetura da máquina.

4.1.3 - Compilação Controlada.

Faz-se a inserção de diretivas de compilação no fonte. Assim, o usuário indica ao compilador exatamente como cada trecho do programa deve ser compilado. Estas diretivas permitem a especificação de quais trechos do programa devem ser paralelizados, vetorizados, "unrolled", etc.

Utilizando-se diretivas de compilação é possível obtermos o maior desempenho possível do aplicativo. Mas é necessário que se tenha um ótimo conhecimento da arquitetura alvo. Além disto, quando portarmos este fonte para outras máquinas, as diretivas de compilação provavelmente terão de ser modificadas.

LINPACK FORTRAN (vetores de 100 elemen.).
130.2 mflops (vetorizando)
142.3 mflops (paralelizando)
LINPACK C (vetores de 100 elementos)
61.8 mflops (vet. ou paralel.)
LINPACK FORTRAN (vetores de 1000 elem.).
291.2 mflops (vetorizando)
328.1 mflops (paralelizando).

Tabela 1. Resultados do Linpack no Cray.

A influência do compilador sobre o desempenho da aplicação pode ser comprovado quando comparamos o desempenho obtido pelo LINPACK codificado em FORTRAN e pelo LINPACK codificado em C (Tabela 1). O algoritmo implementado nas versões C e FORTRAN é o mesmo. No caso do CRAY, quando a otimização é feita automaticamente pelo compilador, sem auxílio do usuário, o compilador FORTRAN gera um aplicativo que roda quase duas vezes mais rápido que quando compilado em C. Isto não significa que, utilizando o compilador C, não podemos gerar um aplicativo

que obtenha um alto desempenho. Inserindo-se diretivas de compilação no código, obtem-se código bem otimizado. Mas, no modo automático, o compilador FORTRAN gera um código mais otimizado.

The NAS Kernel.
Apenas vetorizado (-Zv).

KERNEL	FP OPS	SEC.	MFLOPS
MXM	4.19E+08	1.5077	278.19
CFFT2D	4.98E+08	6.6176	75.27
CHOLSKY	2.21E+08	2.5107	88.04
BTRIX	3.22E+08	2.7243	118.19
GMTRY	2.27E+08	0.9186	246.58
EMIT	2.26E+08	1.2446	181.63
VPENTA	2.59E+08	5.06	51.27
TOTAL	2.17E+09	20.5835	105.5442

Tab. 2. Desempenho do NAS kernel vetorizado no Cray Y-MP/2E.

A eficiência do vetorizador do compilador FORTRAN do Cray pode ser notada através dos benchmarks realizados. Podemos notar que este consegue otimizar bem o código, automaticamente, sem a inserção de diretivas de compilação no fonte. Mas o paralelizador não possue a mesma eficiência. Apenas quando inserimos diretivas de compilação, indicando onde o código deve ser paralelizado, conseguimos quebrar a barreira dos 333 MFLOPS (desempenho máximo de UM processador do Cray Y-MP/2E).

O desempenho insatisfatório do paralelizador pode ser observado claramente através das Tabela 1, 2 e 3. Nelas notamos que nenhum dos programas de benchmark obteve incrementos significativos no seu desempenho quando a paralelização automática foi requisitada.

O desempenho insatisfatório do paralelizador fica ainda mais evidente quando comparamos o desempenho obtido pelos programas EXEMPLO1 e EXEMPLO2. Os dois programas são idênticos. A única diferença reside no fato de que no programa EXEMPLO2 inserimos diretivas de compilação, sendo que somente assim conseguimos a paralelização. Notamos que com o programa EXEMPLO1, compilado sem diretivas de compilção no fonte, obtemos um desempenho de 324.01 MFLOPS, sendo que quando inserimos diretivas de compilação no fonte (EXEMPLO2), atingimos um desempenho de 649.66. O trecho onde é realizada a avaliação da performance, no programa EXEMPLO1, está reportado no ítem 3.2. No EXEMPLO2 inserimos diretivas selecionando o laço mais externo para ser paralelizado e o laço mais interno para ser vetorizado.

The NAS Kernel.
Vetorizado e Paralel. (-Zp).

KERNEL	FP OPS	SEC.	MFLOPS
MXM	4.19E+08	1.4857	282.3114
CFFT2D	4.98E+08	6.6074	75.38063
CHOLSKY	2.21E+08	2.3787	92.9205
BTRIX	3.22E+08	3.076	104.6717
GMTRY	2.27E+08	0.934	242.5054
EMIT	2.26E+08	1.2456	181.4708
VPENTA	2.59E+08	4.5638	56.84517
TOTAL	2.17E+09	20.2912	107.0646

Tab. 3. Desempenho do NAS kernel paralelizado no Cray.

A ineficiência do paralelizador também pode ser notada quando observamos o "Job Accounting" gerado pelo UNICOS (sist. operacional do Cray). No "Job Accounting", notamos que do tempo necessário ao processamento do *Job*, grande parte é consitituido por um intervalo de tempo em que apenas uma das CPU's está destinada a este. Como no exemplo da Fig. 1, quando durante os 20.8169 seg. necessários a execução do *Job*, apenas durante 1.3676 seg. as duas CPU's estavam destinadas a este. Estes testes foram realizados quando não havia nenhum outro *Job* submetido.

```
Job Accounting - Summary Report
===============================

Concurrent CPUs * Connect seconds = CPU sec.
--------------- --------------- -----------
1 *      19.4492 =    19.4492
2 *       1.3676 =     2.7353

Concurrent CPUs * Connect seconds = CPU sec.
  (Avg.)          (total)         (total)
---------------   --------------- -----------
    1.07    *      20.8169   =    22.1845
```

Fig. 1. Trecho do Job Accounting do Cray.

A influência do compilador sobre o desempenho do programa varia de máquina para máquina, como pode ser observado quando medimos o desempenho obtido nas estações de trabalho SUN e em PC's, utilizando os compiladores C ou FORTRAN. O desempenho obtido quando compilamos o bechmark LINPACK em C e FORTRAN, foi o mesmo. Isto já poderia ser esperado, visto que estas máquinas não possuem unidades vetoriais e possuem apenas um processador. Veja o desempenho das estações de trabalho SUN e dos PCs nas tabelas 6 e 7.

4.2 - Tarefa a ser executada:

A tarefa a ser realizada certamente é um dos fatores que mais influenciam o desempenho do aplicativo. Cada tarefa tem suas peculiaridades, em relação à caracteristica da máquina mais exigida. Por exemplo, existem tarefas que fazem uso intensivo de vetores, outras que possuem extensas áreas de código que podem ser executadas em paralelo, e outras que fazem uso intensivo de I/O ou memória. Assim, a mesma tarefa realizada em três máquinas diferentes, uma vetorial, uma massivamente paralela e outra com bastante memória e dispositivos de I/O extremamente rápidos, todas com o mesmo desempenho máximo teórico, pode obter em desempenho extremanente diferente em cada uma das três máquinas.

LIVERMORE FORTRAN

Maximum Rate	=	289.6052 Mega-Flops/Sec.
Average Rate	=	80.3785 Mega-Flops/Sec.
Geometric Mean	=	45.1993 Mega-Flops/Sec.
Harmonic Mean	=	25.0768 Mega-Flops/Sec.
Minimum Rate	=	3.6370 Mega-Flops/Sec.
Standard Dev.	=	81.2194 Mega-Flops/Sec.

Tabela 4. Resultados Livermore

A influência da tarefa a ser realizada no desempenho, pode ser observada nas Tabelas 1, 2 e 4. Na Tabela 2 notamos que o mesmo programa de benchmark (The NAS kernel) , com diferentes rotinas de teste de desempenho, obtém desempenho diferente nas diferentes rotinas. Estas rotinas foram todas

escritas pela mesma equipe, portanto a qualidade do código, a princípio, é a mesma. A distinção se faz na tarefa realizada. A variação no desempenho obtido é grande, de 51.27 MFLOPS a 278.19 MFLOPS. Devemos ressaltar ainda que as tarefas realizadas são semelhantes (resolução de problemas numéricos). Sabe-se também que um mesmo programa (LINPACK FORTRAN) obtém desempenhos muito diferentes quando variamos o tamanho do problema a ser resolvido, isto é, modificamos apenas o tamnaho dos vetores (de 100 para 1000 elementos), como pode ser visto na tabela 1. Na Tabela 4 também podemos observar a grande variação de desempenho obtido em diferentes partes do programa de benchmark LIVERMORE. Portanto, variações de desempenho ainda maiores podem ser esperadas quando da implementação da resolução de outros problemas. Estas variações de desempenho ainda maiores, devido a tarefa a ser realizada, puderam ser observadas quando da implementação do algoritmo "Quick Sorting Paralelo" no Cray, por Teixeira Jr [2]. Neste caso, o desempenho obtido foi inferior a 5 MFLOPS. Na implementação utilizada deste algoritmo não se fez uso de vetores.

PROGRAM	FP OPS	SECONDS	MFLOPS
EXEMPLO2	1.677E+09	2.5153	649.66
EXEMPLO1	1.677E+09	5.1778	324.01
EXEMPLO3	1.677E+09	15.9246	105.35

Tabela 5. Influência do Tunning.

4.3 - Tunning:

A "Sintonia" do aplicativo com a máquina é fundamental. O aplicativo deve explorar ao máximo a característica mais importante da máquina em questão (como vetorização, paralelismo, etc.), não sendo importante a otimização do código em relação a outros aspectos (por exemplo, é inútil escrever o código de maneira paralelizável em máquinas vetorias com apenas um processador). É importante que sejam obeservados todos os detalhes da arquitetura/compiladores para obter-se esta sintonia, como por exemplo no CRAY, deve-se observar a ordem em que as variáveis de controle dos laços aparecem na indexação dos vetores no interior do laço, para permitir a máxima utilização das unidades vetoriais. O impacto da escolha das variáveis de controle dos laços pode ser observado na Tabela 5. Esta escolha é importante devido às contenções que podem ocorrer ao acessar-se seqüêncialmente um mesmo banco de memória. Verificamos que quando executamos o programa EXEMPLO3, que é o programa EXEMPLO1 modificado para forçar a ocorrência de contenções no acesso à memória, escolhendo-se inadequadamente a variável de controle do laço mais interno, o desempenho cai drasticamente.

Máquina:	**Desempenho:**
SPARC 330 Server =	2.38 mflops
SPARC station 1+ =	1.72 mflops
SPARC station IPC =	1.65 mflops
SPARC station SLC =	0.98 mflops

Tabela 6. Desempenho das SUNs (Linpack 100 elem.).

Máquina:	**Desempenho:**
386DX 33 Mhz c/ 387:	0.31 mflops
286 16 Mhz s/ 287 :	0.08 mflops

Tabela 7. Desempenho dos PCs (Linpack 100 elem.).

A influência da tarefa a ser executada e da sintonia do aplicativo com a máquina também pode ser observada, como contra-exemplo, quando realizou-se a comparação do desempenho obtido durante a simulação elétrica, utilizando-se o Cray Y-MP/2E. Simulou-se o ciclo completo (carga, execução da

operação e escrita do resultados) de uma ULA, implementada em uma tecnologia CMOS de 1.2 µm de comprimento de canal mínimo. Usou-se um simulador elétrico de uso geral (SPICE3-C no Cray e PSPICE v.4.02 em um PC386 SX, clock de 25 MHz, com coprocessador 387). A ULA está dividida em *bit slices*. Realizou-se quatro simulações transientes. A diferença entre elas reside no número de *bit slices* utilizados e no espaço de tempo abrangido. Realizou-se simulações com *bit slices* de 2 e 4 bits, abrangendo-se um periodo de 960 e 1920 segundos. Em todas as simulações o passo (*step*) de incremento do tempo simulado utilizado foi de 1 ns.

Os resultados estão na tabela 8.

Tam. Circuito	Tempo Analisado	Cray-YMP/2E	PC-386	Ganho
2 bits	0 - 960 ns.	16.48 s.	293.58 s.	17.81
4 bits	0 - 960 ns.	40.82 s.	719.90 s.	17.63
2 bits	0 - 1920 ns.	38.21 s.	650.58 s.	17.01
4 bits	0 - 1920 ns.	93.18 s.	1650.91 s.	17.23

Tab. 8. Comparação de Resultado de Simulação Elétrica (Spice).

Como podemos observar, o ganho de desempenho obtido no Cray-YMP/2E não foi proprocional a diferença das potencialidades das máquinas utilizadas. Isto ocorreu porque o código do simulador utilizado (SPICE) é altamente escalar. O tamanho médio do vetor armazenado nas unidades vetoriais do Cray durante as simulações foi de 2 elementos. É também possível observar que o ganho obtido varia de acordo com o tamanho do problema. O mesmo foi observado durante a utilização de programas simuladores de propóstio geral (ANSYS, por exemplo) para análise numérica utilizando-se o método de elementos finitos.

5 - COMPARANDO MEDIDAS DE DESEMPENHO.

Existem várias maneiras de relatar medidas de desempenho que possuem validade duvidosa como instrumentos de comparação entre diferentes arquiteturas.

Os seguintes fatores podem afetar o grau de confiança nas comparações de desempenho entre máquinas:

5.1 - Medir desempenho apenas em operações de 32 bits, em máquinas que possuem palavras de 32 bits. Em supercomputação, subentende-se que o desempenho foi medida em MFLOPS com números de 64 bits. Muitos fabricantes de máquinas de 32 bits realizam benchmarks com números de 32 bits, e comparam seus resultados com os obtidos com máquinas de 64 bits.

5.2 - Silenciosamente empregar código em assembler, ou outras construções de baixo nível. Como vimos na seção 2, um dos fatores de grande impacto no desempenho é o compilador. Assim, não tem sentido comparar-se o desempenho obtida quando emprega-se código de baixo nível em um máquina com o desempenho de código de alto nível de outra máquina.

5.3 - O desempenho obtido com um kernel interno facilmente vetorizável ou paralelizável é maior do que o desempenho obtida por todo o programa. É necessário clareza na comparação dos resultados obtidos a partir de kernels internos ou programas.

5.4 - Em arquiteturas multiprocessadas, redimensionar o tamanho do problema, quando aumenta o número de processadores empregados.

5.5 - Comparar o desempenho obtido por programas diferentes em máquinas diferentes. Não tem sentido, por exemplo, rodar um aplicativo A, com código antigo e mal otimizado, numa máquina M, e rodar um aplicativo B, com código otimizado, numa máquina N, a fim de comparar o desempenho da máquina N com a máquina M.

5.6 - Utilizar aplicativos extremamente curtos, onde um número de instruções muito pequeno é executado durante o benchmark. Nas máquinas vetoriais e/ou paralelas, tarefas muito pequenas não preenchem o pipeline vetorial e/ou não são paralelizadas. Assim, o desempenho destas máquina pode ficar muito parecido com o de máquinas mais simples, quando tarefas pequenas são executadas.

5.7 - Cotar o desempenho em MFLOPS por dolar (MFLOPS/$). Esta medida não é muito útil, porque como vimos anteriormente, a relação tamanho do sistema/desempenho não é linear. Além disto, existem grandes problemas em colocar-se um número maior de processadores e/ou máquinas para resolver o mesmo problema. O tempo necessário a resolução de um determinado problema também é importante, não apenas o custo dispendido em resolvê-lo.

6 - CONCLUSÃO.

Através dos benchmarks realizados foi possível verificar na prática a influência de diversos fatores sobre o desempenho obtido por uma máquina.

Assim, quando o desempenho de uma determinada arquitetura é citado, devemos sempre levar em consideração o tipo de problema resolvido, o compilador utilizado, e se o código foi escrito especificamente para a arquitetura utilizada. Estes fatores são extremamente relevantes quando precisamos optar entre diferentes sistemas, e utilizamos valores de desempenho, publicados por fabricantes ou outras fontes, para fazer esta opção.

Certamente a medida de desempenho mais significativa quando da escolha de uma máquina no momento da aquisição, é a medida do tempo de execução do problema a ser resolvido (não confunda com o desempenho relatado pelos programas que monitoram o desempenho, citados no ítem 2.1, porque estes geralmente não computam o tempo gasto em I/O, computam apenas o desempenho da CPU). Esta é a melhor medida, porque o mais importante é conseguirmos executar as tarefas no menor tempo possível, tirando-se assim o máximo proveito da máquina.

Mas, na prática, uma máquina dificilmente é utilizada para executar sempre uma mesma tarefa. Então, faz-se necessário conhecer as características comuns à maioria dos programas a serem utilizados, e analisar o adequação das máquinas a estas caracteristicas, tendo-se em mente os tópicos descritos no decorrer do presente trabalho.

7 - REFERÊNCIAS.

[1] - Weicker, Reinhold P. "A detailed look at some popular benchmarks". Parallel Computing, Vol. 17, Numbers 10&11, December 1991, p. 1153-1172.

[2] - Teixeira Jr., Carlos A. "Algoritmo Quicksort Paralelo". Trabalho da Disciplina de Arquitetura de Computadores para Processamento Paralelo, CPGCC-UFRGS, Porto Alegre, julho de 1993.

[3] - D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS Parallel Benchmarks", Intl. Journal of Supercomputer Applications, v. 5, no. 3 (Fall 1991), pp. 63-73.

[4] - Weicker, Reinhold P. "Dhrystone benchmark: Rationale for Version 2 and measurement rules". SIGPLAN Notices 23 (8), August 1988, pp. 49-62.

[5] - Currow, H.J. and Wichmann, B. A. "A Synthetic benchmark" Comput. Journal, 19 (1), 1976, pp. 43-49.

[6] - Wichmann, B. A. "Validation Code for the Whetstone Benchmark", Report NPL-DITC 107/88, National Physical Laboratory, Teddington, U.K., March 1988, 16 p.

[7] - Dongarra, J.J. et al., "LINPACK User's Guide". SIAM Publications, Philadelphia, PA, 1976.

[8] - Dongarra, J.J. "Performance of Various Computers Using Standard Linear Equations Software". ACM Computers and Architecture News , Vol. 18, Iss. 3, June 1992, pp. 22-44.

[9] - McMahon, F. H. "The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range", Lawrence Livermore National Laboratory, Livermore (CA), Technical Report UCRL-53745, Dec. 1986, 179 p.

[10] - McMahon, F. H. "The Livermore Fortran Kernels Test of the Numerical Performance Range". In: J.L. Martin, ed. "Performance Evaluation of Supercomputers". North-Holland", Amsterdam, 1988, pp. 143-186.

[11] - M. Berry, G. Cybenko and J. Larson. "Scientific Benchmark Characterizations". Parallel Computing, Vol. 17, Numbers 10&11, December 1991, p. 1173-1194.

[12] - Uebel, Luis F. "Bechmark". Trabalho da Disciplina de Arquitetura de Computadores para Processamento Paralelo, CPGCC-UFRGS, Porto Alegre, agosto de 1992.