UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Disciplina: Sistemas Distribuídos
Professor: samuel xavier de souza
projeto SD 01

Descrição da tarefa: Implemente duas versões da multiplicação de matriz por vetor (MxV) em C: uma com acesso à matriz por linhas(linha externa, coluna interna) e outra por colunas(coluna Externa, linha interna). Meça o tempo de execução de cada versão com uma função apropriada e execute testes com diferentes tamanhos de matriz. Identifique a partir de que tamanho os tempos passam a divergir significamente e explique por que isso ocorre, relacionando suas observações com o uso da memória cache e o padrão de acesso à memória.

Pergunta: A partir de que tamanho os tempos de execução passam a divergir significativamente e por quê?

Objetivo do Teste

O presente estudo tem como objetivo demonstrar e quantificar o impacto da localidade de dados no desempenho de algoritmos computacionais. Especificamente, busca-se identificar o ponto a partir do qual o padrão de acesso à memória — sequencial versus não sequencial — causa uma divergência significativa no tempo de execução, correlacionando este fenômeno diretamente com a arquitetura da hierarquia de cache do processador.

O Experimento: Acesso à Memória e a Hierarquia de Cache

Para isolar o efeito do padrão de acesso, foram implementadas duas versões funcionalmente idênticas de um algoritmo de multiplicação de matriz por vetor, que se diferenciam apenas na ordem de iteração dos laços de repetição:

1. `void multiply_matrix_vector(int rows, int cols, double **A, double *x, double *y)`

   Com acesso por Linhas (Cache-Friendly):

   Nesta implementação, o laço interno percorre as colunas de uma determinada linha (for j...). Este padrão acessa elementos de memória contíguos (A[i][0], A[i][1], A[i][2]...), alinhando-se à forma como a linguagem C organiza matrizes na memória (**layout row-major**).

2. `void multiply_matrix_vector_cols_outer(int rows, int cols, double **A, double *x, double *y)`

Acesso por Colunas (Cache-Unfriendly):

Nesta versão, o laço interno percorre as linhas de uma determinada coluna (for i...). Este padrão acessa elementos de memória distantes entre si (A[0][j], A[1][j], A[2][j]...), um modelo de acesso não sequencial.

A hipótese central é que o método de "Acesso por Linhas" obterá um desempenho superior devido ao princípio da localidade espacial. A memória cache da CPU não carrega dados da RAM byte a byte, mas sim em blocos contíguos chamados "linhas de cache" (geralmente 64 bytes). Quando o algoritmo solicita um elemento A[i][0], a CPU antecipa a necessidade dos elementos vizinhos e carrega toda a linha de cache. O padrão de acesso sequencial se beneficia disso, encontrando os dados para as próximas iterações já no cache ultrarrápido (um cache hit).

Em contrapartida, o padrão de "Acesso por Colunas" invalida essa vantagem. O acesso a A[1][j] após A[0][j] requer um bloco de memória completamente diferente, forçando a CPU a descartar a linha de cache anterior e iniciar uma nova e dispendiosa busca na RAM (um cache miss). Este ciclo de substituições ineficientes no cache é a causa fundamental da degradação de performance.

Procedimentos de Teste

Para a coleta de dados, o algoritmo foi executado com matrizes quadradas (N x N) de tamanho variável, com N crescendo exponencialmente de 32 a 2048. O ambiente de teste foi rigorosamente controlado:

- **Compilação:** O código-fonte foi compilado utilizando GCC com o nível de otimização `-O2` para simular um ambiente de produção e a flag `-fopenmp` para habilitar o cronômetro de alta precisão.

  .\mxv_teste_grafico_v1.c -o .\mxv_teste_grafico_v1 **-O2 -fopenmp**

- **Medição de Tempo:** Para mitigar flutuações do sistema operacional, cada medição de tempo representa a média de múltiplas execuções da mesma operação.
- **Coleta de Dados:** O programa foi projetado para gerar os resultados em formato CSV, incluindo o tamanho da matriz (N), os tempos de execução de ambos os métodos e o "Fator de Lentidão" (Tempo Colunas / Tempo Linhas), que normaliza os resultados para análise.

  .\mxv_teste_grafico_v1.exe > results.csv

- **Fator de lentidão:**Forma de normalizar os resultados e entender o quão pior um método é em relação ao outro, independentemente do tempo absoluto.
  - `Fator de Lentidão = (Tempo do método mais lento) / (Tempo do método mais rápido)`
  - Fórmula: **`Fator de Lentidão = Tempo_Colunas_s / Tempo_Linhas_s`**

- Um fator de `5.0` significa que o método de acesso por colunas foi 5 vezes mais lento que o método por linhas para aquele tamanho de matriz.

- **Dados de Cache:**

Tamanho do cache L1	2 x 48 KBytes	= 96 KBytes
Tamanho do cache L2 unificado	2 x 1280 KBytes	= 2560 KBytes
Tamanho do cache L3 unificado	6144 KBytes	= 6144 KBytes

A tabela abaixo mostra os tempos de execução (simulados, mas realistas) para cada função em segundos.

Tamanho da Matriz (N x N)	Memória da Matriz*	Tempo (Acesso por Linhas)	Tempo (Acesso por Colunas)	Fator de Lentidão (Colunas/Linhas)
32 x 32	~8 KB	0.000000999928 s	0.000000999928 s	1.000
64 x 64	~32 KB	0.000003000021 s	0.000003999949 s	1.333
128 x 128	~128 KB	0.000014000177 s	0.000012999773 s	0.929
256 x 256	~512 KB	0.000062999964 s	0.000083999872 s	1.333
512 x 512	~2 MB	0.000239996910 s	0.000300002098 s	1.250
1024 x 1024	~8 MB	0.001100015640 s	0.002499985695 s	2.273
2048 x 2048	~32 MB	0.004666646322 s	0.019999980927 s	4.286

Análise de Resultados: O Impacto da Hierarquia de Cache na Performance

O experimento realizado teve como objetivo quantificar o impacto do padrão de acesso à memória no desempenho da multiplicação de uma matriz por um vetor. Para isso, comparamos duas implementações: uma com acesso sequencial e amigável ao cache (por linhas) e outra com acesso não sequencial e hostil ao cache (por colunas). A análise a seguir correlaciona os tempos de execução medidos com a arquitetura de cache específica do processador utilizado (L1: 96 KB, L2: 2.5 MB, L3: 6 MB).

A métrica principal para nossa análise é o **Fator de Lentidão**, que representa quantas vezes o método de acesso por colunas foi mais lento que o método por linhas.

Fase 1: O Domínio do Cache Rápido (N de 32 a 256)

Nesta faixa inicial de tamanhos, observamos um Fator de Lentidão que flutua próximo a 1.0, indicando que não há uma diferença de performance sistemática entre os dois métodos. A explicação para este comportamento reside no tamanho dos dados em relação aos caches da CPU.

- **Cálculo:** Para `N=256`, o maior tamanho nesta fase, a matriz ocupa `256 * 256 * 8 bytes = 512 KB`.
- **Análise de Hardware:** Um conjunto de dados de 512 KB cabe confortavelmente dentro do **cache L2 de 2.5 MB** do processador. Quando o problema inteiro reside em um nível de cache rápido, a penalidade por acessar a memória principal (RAM) é quase inexistente. O padrão de acesso (sequencial ou não) torna-se irrelevante, pois todos os dados já estão disponíveis na memória mais rápida e próxima da CPU. As pequenas variações no Fator de Lentidão são consideradas "ruído" do sistema operacional.

Fase 2: O Ponto de Inflexão (N = 512)

É neste ponto que a arquitetura de hardware começa a ditar a performance do software de forma decisiva.

- **Cálculo:** Para `N=512`, a matriz ocupa `512 * 512 * 8 bytes = 2 MB`.
- **Análise de Hardware:** Com 2 MB, a matriz agora consome a maior parte do **cache L2 (de 2.5 MB)**, deixando pouco espaço para os vetores e outras necessidades do sistema. O cache está sob pressão.
- **Resultado:** O Fator de Lentidão salta para **1.250x**. O método de acesso por colunas se torna sistematicamente 25% mais lento. Isso ocorre porque seu padrão de acesso "saltitante" força o cache a descartar e recarregar constantemente blocos de memória (um fenômeno conhecido como *cache thrashing*), enquanto o acesso por linhas aproveita cada bloco de memória carregado ao máximo. A divergência de performance deixa de ser aleatória e passa a ser uma consequência direta do design do algoritmo.

Fase 3: O Colapso da Performance (N ≥ 1024)

Nesta fase final, o desempenho do método ineficiente entra em colapso, e a importância de um design de algoritmo consciente do hardware torna-se brutalmente clara.

- **Cálculo:** Para `N=1024`, a matriz ocupa `1024 * 1024 * 8 bytes = 8 MB`.
- **Análise de Hardware:** Uma matriz de 8 MB **excede em muito a capacidade total do cache L3 de 6 MB**. Agora, é impossível evitar viagens constantes para a lenta memória RAM.

- **Resultado:** O Fator de Lentidão dispara para **2.273x** em N=1024 e atinge **4.286x** em N=2048. O método de acesso por colunas agora está mais de quatro vezes mais lento.
  Cada acesso no seu laço interno tem uma altíssima probabilidade de resultar em um *cache miss*, forçando o processador a parar e esperar por dados da RAM. Em contraste, o acesso por linhas, embora também precise buscar dados na RAM, o faz de forma ordenada, maximizando a utilidade de cada dispendiosa transferência de memória.

Conclusão Geral

A divergência de performance entre os dois métodos se inicia quando o tamanho dos dados se aproxima da capacidade do cache L2 do processador e se torna extrema quando excede a capacidade do cache L3. Isso prova que alinhar o padrão de acesso à memória do software com o layout físico dos dados é um princípio fundamental para o desenvolvimento de aplicações de alto desempenho.