ParLSS: a Parallel Self-Verified Solver for Dense Systems of Linear Equations

Alexandre Almeida – avalmeida@inf.ufrgs.br
Carlos A. Hölbig – holbig@upf.br

Curso de Ciência da Computação
Universidade de Passo Fundo

WSPPD 2009
VII Workshop de Processamento Paralelo e Distribuído

Roteiro da apresentação

Introdução

Sistemas lineares densos

- Importância para a área da Análise Numérica [Cláudio and Marins, 2000]
  - Análise tensorial de estruturas elétricas
  - Análise de vibrações em sistemas mecânicos
- Fatores críticos: tempo de execução e qualidade numérica

Introdução

Sistemas lineares densos

- Importância para a área da Análise Numérica [Cláudio and Marins, 2000]
  - Análise tensorial de estruturas elétricas
  - Análise de vibrações em sistemas mecânicos
- Fatores críticos: tempo de execução e qualidade numérica

Computação verificada (*Verified computing*)

- Trata a instabilidade numérica de máquina
- Sistema computacional garante a qualidade numérica, ou alerta a não-obtenção do resultado
- Biblioteca de alta exatidão C-XSC (www.xsc.de)

Introdução

Solver LSS

- Resolução de sistemas de equações lineares com matrizes densas
- Computação verificada, com apoio da biblioteca C-XSC
- Alta qualidade numérica
- Baixo desempenho:
  - 7h35min para ordem 2048:
    - Intel Core 2 Duo E4500 @ 2,2 GHz (667 MHz FSB)
    - 2 GB de memória principal

Objetivos

1. Otimizar a versão sequencial do LSS
2. Projetar e implementar uma versão paralela em MPI-1
   - Baseada na versão otimizada
3. Implementar funções de comunicação coletiva com suporte à biblioteca C-XSC
   - Extensão às funções para troca de dados ponto-a-ponto, apresentadas em [Grimmer, 2008]

Solver LSS – Overview

- Programa que tenta encontrar uma solução dado um sistema linear denso:

$$Ax = b, \text{ onde:}$$

- Entrada: matriz densa de coeficientes A; vetor de termos independentes b
- Saída: vetor x
- Composto por duas fases de processamento:
  - Etapa de precisão simples
  - Etapa de precisão dupla

Etapas do processamento

- Problema: resolver $Ax = b$

Etapa de precisão simples

- Calcula A^{-1} através de Gauss-Jordan
- Determina x' aproximado: $x' = A^{-1} \times b$
- Refina x' através de estimativas de erro
- Caso não encontre a resposta, executa a etapa de dupla precisão

Etapas do processamento

- Problema: resolver $Ax = b$

Etapa de precisão simples

- Calcula A^{-1} através de Gauss-Jordan
- Determina x' aproximado: $x' = A^{-1} \times b$
- Refina x' através de estimativas de erro
- Caso não encontre a resposta, executa a etapa de dupla precisão

Etapa de precisão dupla

- Efetua o Mecanismo de Rump [Hölbig and Krämer, 2003]
  - Matriz inversa e de resíduos: Ai^{-1} e Ea
- Executa os mesmos procedimentos da etapa de precisão simples, porém considerando sempre Ai^{-1} e Ea
- Maior *overhead*

LSS sequencial otimizado

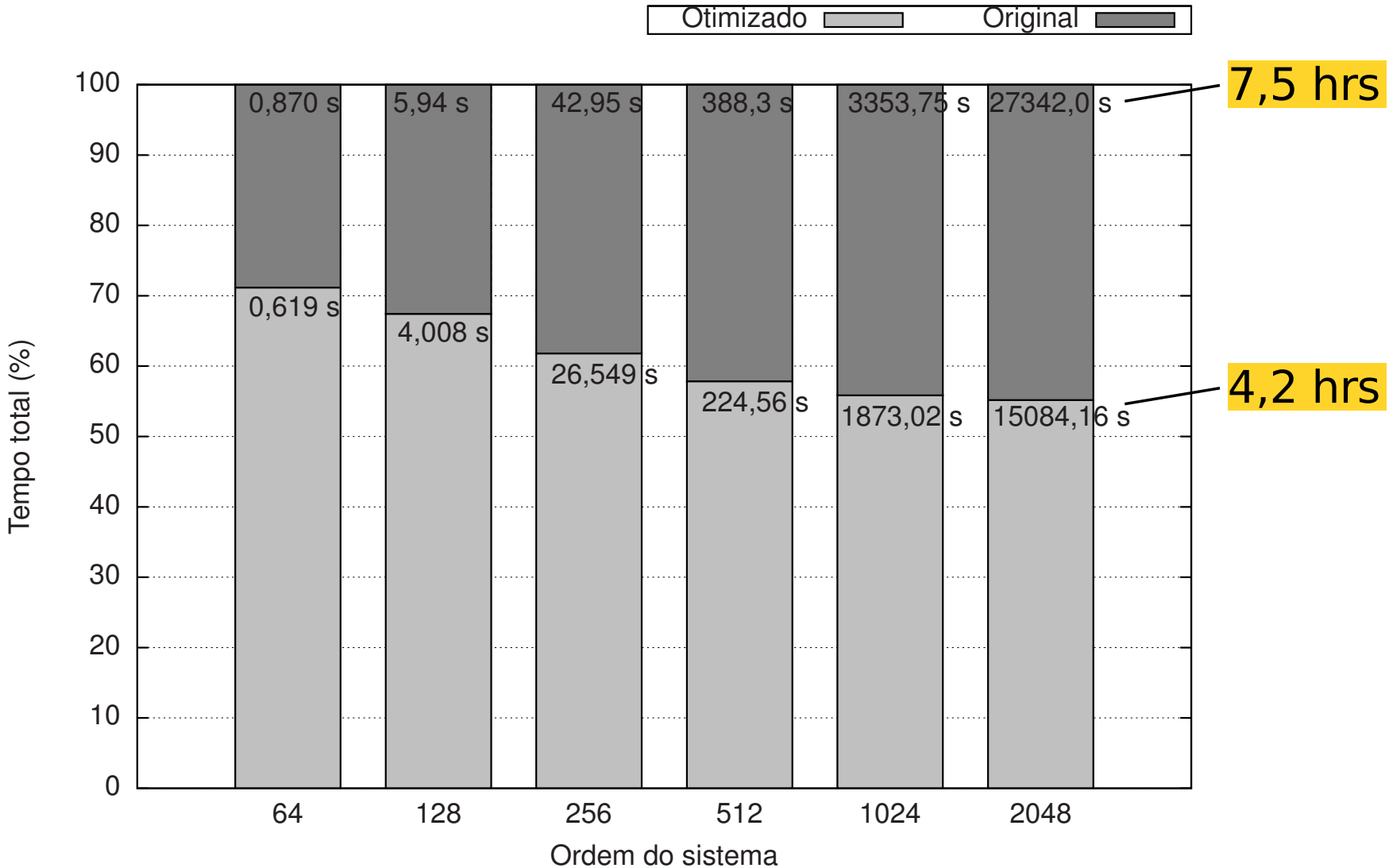
- Obter o "melhor programa" sequencial:
  - Análise de desempenho da versão paralela [Pacheco, 1997]
- Otimizações efetuadas:
  - Eliminação de procedimentos redundantes
  - Otimização do método de Gauss-Jordan

Tempos de processamento sequencial – Metodologia

- Intel Core 2 Duo E4500 @ 2,2 GHz
- 2 GB de mem. principal
- Linux kernel 2.6.21 (Ubuntu 7.04)
- 10 execuções

- Matriz de entrada: $H_{i,j} = \frac{1}{i+j-1}$

Tempos de processamento



Projeto da versão paralela

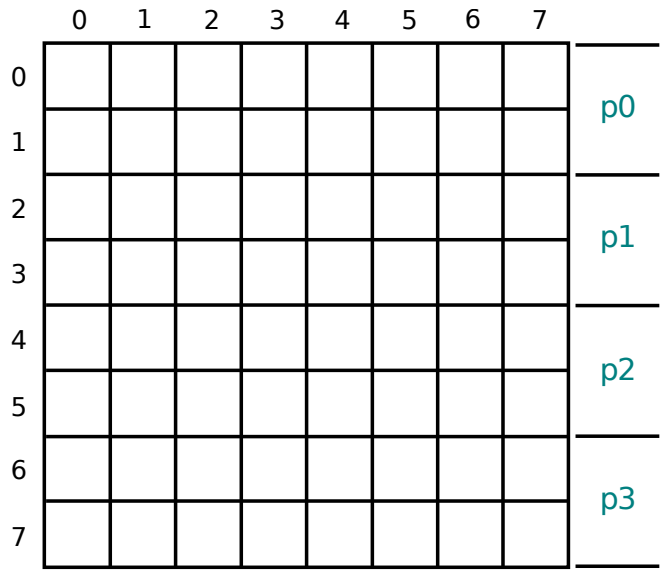
- Baixo desempenho causado por uma pequena porção das sub-rotinas do *solver*

- Na etapa de precisão simples:
  - Inversão de matrizes por Gauss-Jordan: $\approx$ 10 min.
  - $[C] = \diamond\left(I - A^{-1} \times A\right)$: $\approx$ 57 min.
- Na etapa de precisão dupla:
  - Gauss-Jordan: mesma rotina da etapa anterior
  - $Ai^{-1} = \diamond\left(I^{-1} \times A^{-1}\right)$: $\approx$ 56 min.
  - $[C] = \diamond\left(I - Ai^{-1} \times A - Ea \times A\right)$: $\approx$ 2 horas

- *Obs.: tempos para ordem 2048*

Projeto da versão paralela

- Particionamento por blocos de linhas:



- Desenvolvidas funções MPI de comunicação coletiva
  - Suporte aos tipos *CXSC::rmatrix* e *CXSC::imatrix*
  - MPI_Bcast, Gather, Scatter, Allgather

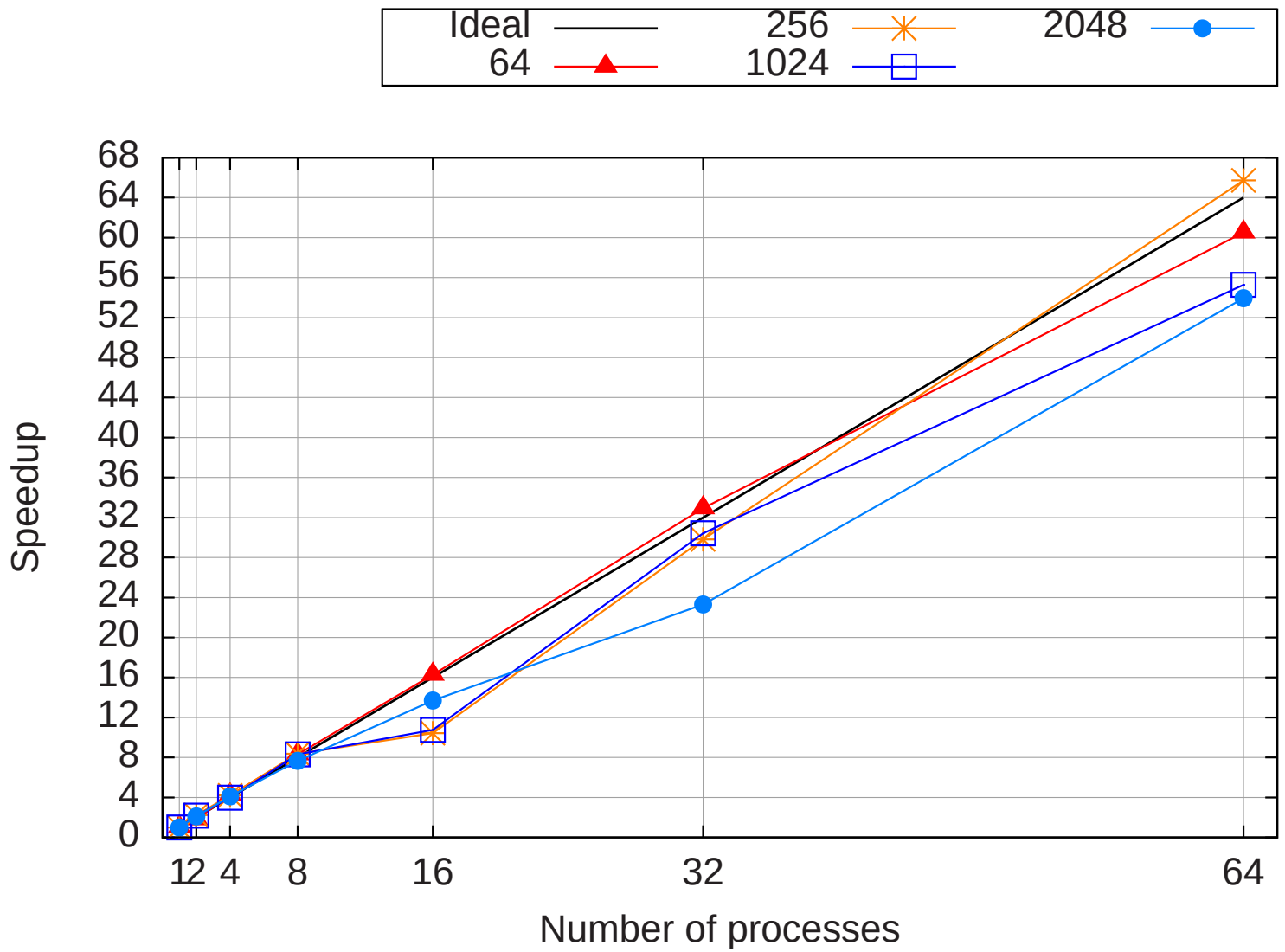
Resultados experimentais – Metodologia

- Network of Workstations:
  - 32 nodos: Intel Core 2 Duo E4500 @ 2.2 GHz
  - 2 GB de memória (2×1 GB *dual-channel*)
  - Rede Fast Ethernet (100 Mbps)
- Linux kernel 2.6.21
- LAM/MPI 7.1.2 – gcc 4.1

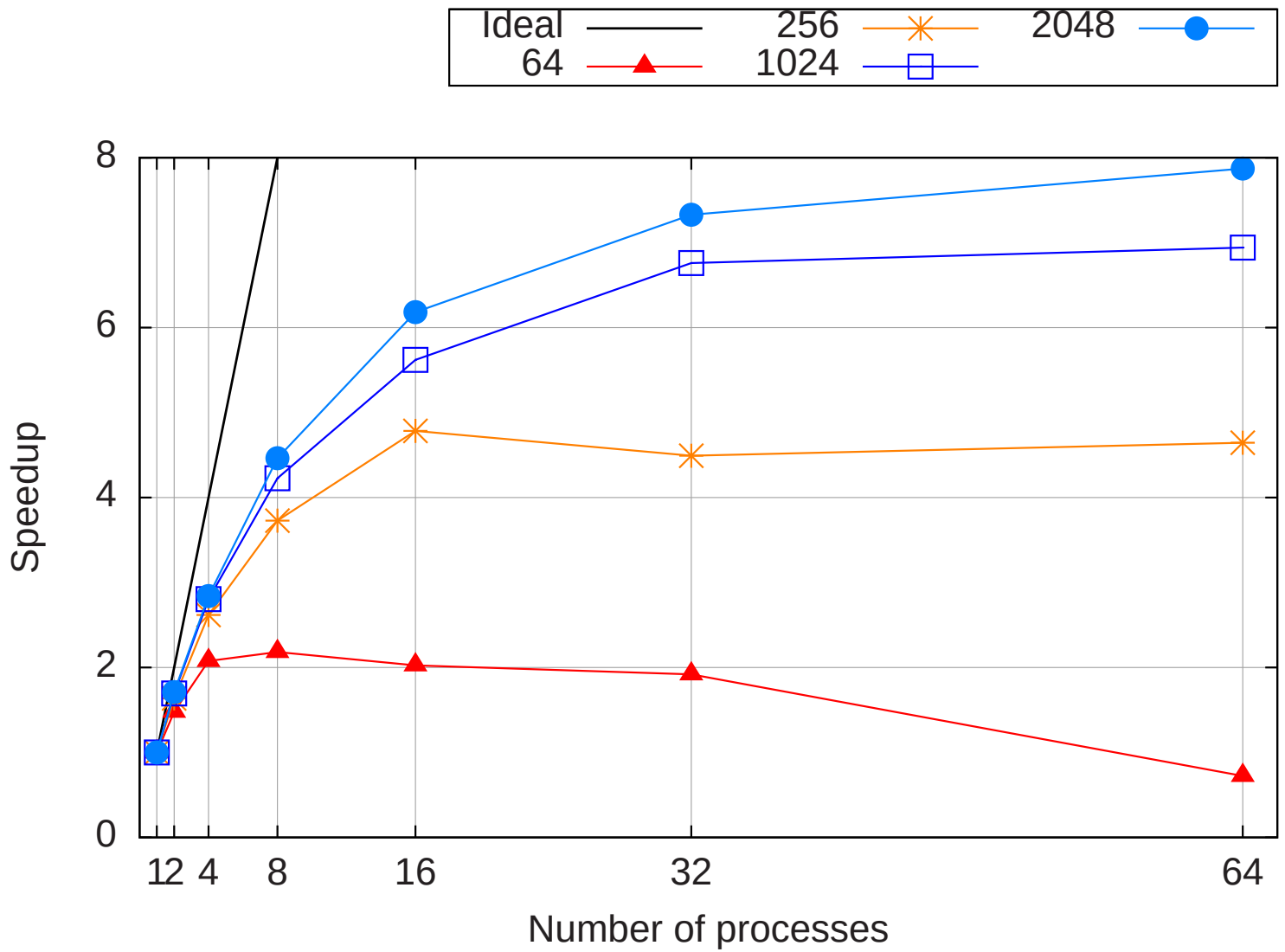
Resultados experimentais – ParLSS

- Sistemas lineares de ordem 64, 256, 1024 e 2048:
  - Matrizes de entrada: $H_{i,j} = \frac{1}{i + j - 1}$
- Execução com 2, 4, 8, 16, 32 e 64 processos:
  - 2 processos por nodo
- Coleta dos tempos de processamento:
  - 10 amostras: desvio padrão abaixo de 2%
  - Tempo de execução de cada trecho paralelo e do *solver* como um todo

Resultados experimentais – ParLSS



Resultados experimentais – ParLSS



Considerações finais

- Considerável ganho em desempenho na versão sequencial
- Baixo *speed-up* da versão paralela:
  - Inversão de matrizes sequencial
  - A distribuição do domínio não é a mais eficiente
- Funções de comunicação coletiva:
  - Redução considerável no custo da comunicação
  - Extensão ao conjunto de funções de [Grimmer, 2008]

Referências

Cláudio, D. M. and Marins, J. M. (2000).

Cálculo Numérico Computacional: Teoria e Prática.

Editora Atlas.

Grimmer, M. (2008).

Extending the range of c-xsc: Some tools and applications for the use in parallel and other environments.

In *Numerical Validation in Current Hardware Architectures*, Dagstuhl Seminar Proceedings, Dagstuhl, Germany. IBFI, Schloss Dagstuhl, Germany.

Hölbig, C. and Krämer, W. (2003).

Selfverifying solvers for dense systems of linear equations realized in c-xsc.

Technical report, Universität Wuppertal.

Pacheco, P. S. (1997).

Parallel Programming with MPI.

Morgan Kaufmann.

ParLSS: a Parallel Self-Verified Solver for Dense Systems of Linear Equations

Perguntas?

Alexandre Almeida – avalmeida@inf.ufrgs.br
Carlos A. Hölbig – holbig@upf.br

Curso de Ciência da Computação
Universidade de Passo Fundo