JPL

# The Matpar Server on the HP Exemplar

**Paul Springer**
**High Performance Computing Group**
**Jet Propulsion Laboratory**
**California Institute of Technology**
**pls@volcanoes.jpl.nasa.gov**
**http://www-hpc.jpl.nasa.gov/PS/MATPAR**

JPL

# Matpar Introduction

Background

Matpar Architecture

Matpar Design

Current Status

PVM vs MPI

Matpar Timings

Conclusions

Future Work

# Background

**Original request from Space Interferometry Mission (SIM) design engineers**

- Matlab too slow on large problems (2000 x 2000 size matrices)
- Many jobs had to be run overnight

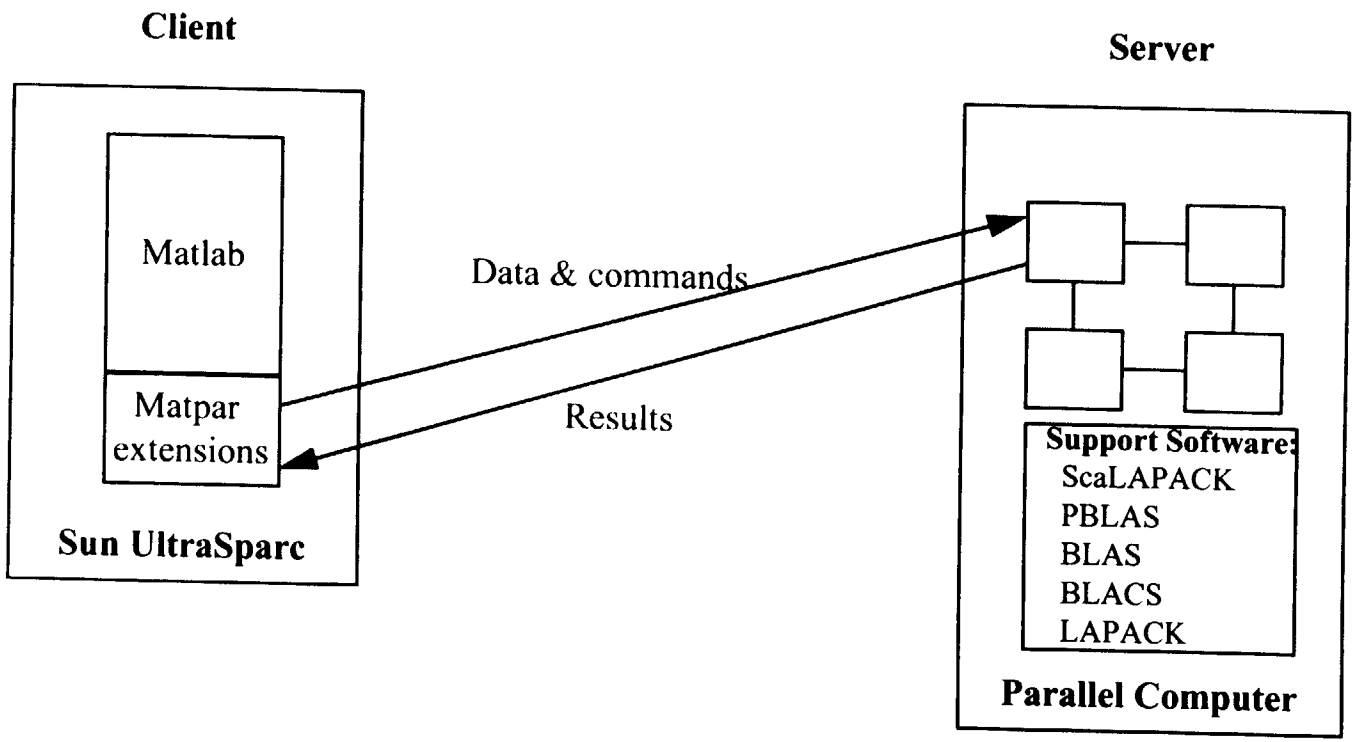
**Next Generation Space Telescope (NGST) needs even greater capability**

- Optical design for segmented mirror
- May use 100,000 x 1,000 size matrices

**Work begun in early 1996 on parallel extensions to Matlab, called Matpar**

Space Interferometry Mission

# Matpar Architecture



# Current Status (1.1 Release)

**Functionality**

- Matrix-matrix addition, subtraction, multiplication: p_add(A,B), p_sub(A,B), p_mult(A,B)
- Scalar multiplication: p_smult(s,A)
- Identity matrix generation: p_eye(m,n)
- Inverse, pseudo inverse: p_inv(A), p_pinv(A)
- SVD: p_svd(A)
- Calculate $A * A^T$ : p_multtrans(A)
- Trace of $A * A^T$ : p_trace(A,1)
- QR factorization: p_qr(A)
- LU factorization: p_lu(A)
- Solve A*X = B: p_solve(A,B)
- Frequency response calculations: p_freqresp(A,B,C,D,w)
- Bode plot calculations: p_bode(A,B,C,D,w)
- Select computer and node count: p_config(computer, node count)
- Persistence: p_persist(A,1)

# Matpar Design

**Only certain operations parallelized**

**Simple Matlab style function calls**

- B=qr(A) becomes B=p_qr(A)
- C=A*B becomes C=p_mult(A,B)

**Calls can be made seamlessly**

**Calls invoke .m and .mex files**

- Matlab standard method of extending the language

**Compatibility with MATLAB versions 4 and 5**

# Client/Server Details

**One MEX file for each parallel routine**

- All MEX files call single shared object module
- All client code written in C

**Matpar client code uses PVM to initiate server session on MPP**

- Server code is object oriented, written in C++
- MPP node 0 is "coordinator" node and communicates to client
- Coordinator node distributes client data to other MPP nodes
- Coordinator node collects result data and sends to client
- Results can be made persistent for one operation, or more permanently
- Matrices larger than 512 x 512 are sent as separate PVM messages
- Server session remains active until user quits MATLAB

# Client/Server Communication Protocol: Requests

**Command (int)**

**Command data: count, data,... (ints)**

- useful for hints, auxiliary command data, etc.

**Data types: count, type,... (ints)**

- IMDBLBLK--double precision matrix sent block by block
- IMDBLDIAG--double precision diagonal matrix
- IMDBLREPMAT-- double precision matrix to be replicated on each node
- REFMAT--reference number for persistent matrix
- IMDBL--double precision scalar
- etc.

**Data: count, data,...**

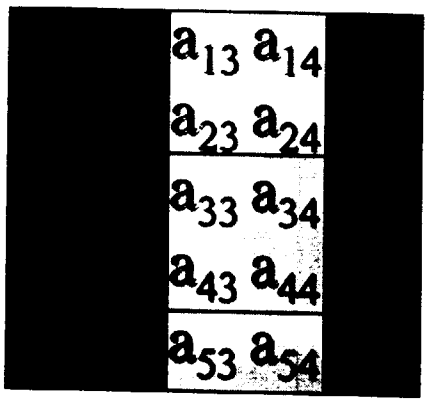
- IMDBLBLK--matrix row count, matrix col count, block row count, block col count, row blocking factor, col blocking factor, data
- IMDBLMAT--row count, col count, data
- REFMAT--reference number

# Client/Server Communication Protocol: Results

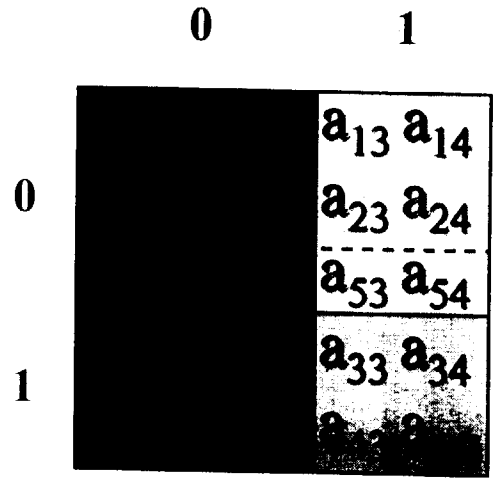
**Data: count, type, data, type, data, ...**

- IMDBL--double precision scalar
- IMDBLMAT-- double precision matrix
- IMDBLCMAT-- double precision matrix with complex data
- IMDBLBLK-- double precision matrix returned block by block
- REFMATSIZE--size of persistent matrix
- IMDBLDIAG-- double precision diagonal matrix

# Block Cyclic Data Distribution




5 x 5 matrix partitioned into
2 x 2 blocks




Matrix distribution onto
2 x 2 node grid

# Matpar Server Objects: Matrices

**Derived classes**

- CompMatrix--complex matrix
- BCompMatrix--banded complex matrix
- RDMatrix--replicated diagonal matrix

**Class data**

- matrix layout
- row and col counts

**Methods**

- constructors & destructors
- replication, collection & distribution
- ScaLAPACK operations
- assignment, equality testing, filling

# Matpar Server Objects: Request

**Class data**

- command
- input data arrays
- result data
- error information

**Methods**

- high level functional calls
- argument retrieval functions
- persistent matrix creation and retrieval
- result creation routines

# Matpar Server Objects: Input Data Items

**Abstract base class to hold each data object**

**Derived classes**

- ImBlkData--immediate block data
- ImDblData--double precision scalar
- ImIntData--integer scalar
- ImMatData--double precision matrix
- ImRepMatData--replicated real matrix
- ImRepMatIData--imaginary part of ImRepMatData
- ImFileData--data from disk file
- RefMatData--reference to persistent matrix

**Class data**

- metadata, pointers to data

**Methods**

- build()
- pack() & unpack() (PVM)

# Matpar Server Objects: Output Data Items

**Abstract base class to hold each data object**

**Derived classes**

- DstIntData--integer scalar result
- DstDblData--double precision scalar result
- DstMatData--matrix result
- DstDiagData--diagonal matrix result

**Class data**

- metadata
- pointers to data

**Methods**

- output() (debugging)
- pack() (PVM)
- packBlock() (PVM)

# Matpar Server Objects: Matrix Block Iterator

**Modeled after C++ Standard Template Library iterators**

**Encapsulates data layout information for this node**

**Allows programmer to efficiently access contiguous data within block**

**Class data**

- global column # of first column of current iterator block
- global row # of first row of current iterator block
- private data regarding block layout

**Methods**

- ++ operator iterates through matrix blocks on this node
- getLastBlkCol()--returns global column # of last column of current block
- getLastBlkRow()--returns global row # of last row of current block

# PVM vs MPI

**Job control requirements**

- Client initiates server transparently to user
- This generally involves starting a script on server, that goes through server's queuing system to start pvmd
- At run time, user decides which of many possible servers to use

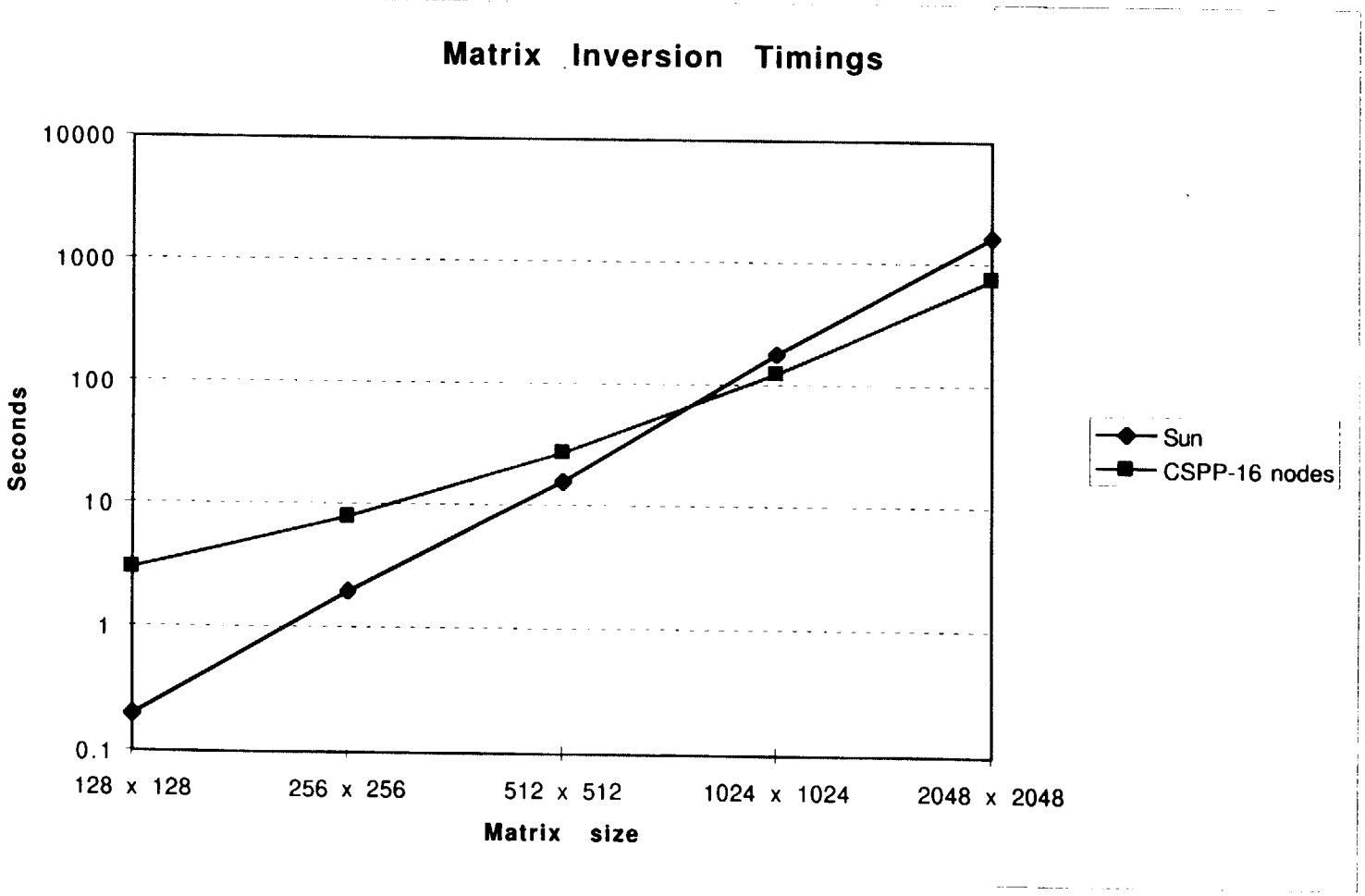
**MPI limitations**

- LAM MPI does not work on our system

  *"recon" reads a blank system table*
- Other versions of MPI do not support dynamic processes
- From MPI-2 spec, p. 86: "If the program named in **command** does not call MPI_INIT, but instead forks a process that calls MPI_INIT, the results are undefined."
- p. 83: "Complex interaction of an MPI application with its runtime environment should be done through ... pvm_addhosts..."
- Workarounds to MPI limitations are cumbersome

JPL

# Timings--Matrix inversion



# Conclusions

**Applying parallelism to MATLAB can speed solutions of large problems**

**The client/server approach Matpar uses provides ease of use and good speed**

**For $O(n^3)$ calculations on large problems, Matpar is faster than MATLAB on a Sun UltraSparc**

- When data transfer rates are included, crossover point is with 512 x 512 matrices
- When no large data transfers are needed (eg. data is persistent), Matpar is faster for even smaller matrices

# Future Matpar Work

**Complete release of version 1.1**

**Port to MPI**

**Completely handle complex matrices**

**Update to use ScaLAPACK 1.5 (matrix redistribution)**

**Do more timings to characterize performance**

- How to determine best block size?
- How to determine ideal number of processors?

# Acknowledgments

The work described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration.

Part of the research reported here was performed using the HP SPP-2000 operated by the Center for Advanced Computing Research at Caltech; access to this facility was provided by Caltech.