MPMPLAPACK: A Massively Parallel Multi-Precision Linear Algebra Package

Jason Martin

jason.worth.martin@gmail.com

James Madison University

Something slightly off topic

Assembly code GMP optimizations for Core 2 Intel chips (Xeon, Core 2 Duo, Core 2 Extreme, etc.) available on my webpage.

`http://www.math.jmu.edu/~martin`

Works on Mac OS X and Linux.

Motivations

- Computer engineer in former life.
- Number theorist now.
- Current research in improving bounds on discriminants of number fields.
- Research hit a road block: exhausted available computational resources.

Motivations

- Computations in much of number theory reduce to linear algebra over Dedekind domains.
- Also occurs in algebraic geometry.
- Applied mathematicians have lots of powerful parallel linear algebra tools.
- Where are the pure math parallel linear algebra tools?

What I Think I Want

An open source software library that:

- Handles linear algebra for modules over “nice” rings.
- Can reasonably use all the cores in my desktop.
- Can reasonably scale to use clusters or supercomputers.
- Can be dropped in to my favorite programs.
- Is very portable.
- Is very very very maintainable.

Linear Algebra

By linear algebra I mean:

- Operations on/with elements, sub-block, rows/coloumns, etc.
- Basic linear algebra (BLAS-ish) routines appropriate to the module and ring.
- Higher-level linear algebra (HNF, SNF, JCF, RCF, etc) when possible

Linear Algebra

By linear algebra I also mean:

- Lattice reduction, Discriminants, Differents, etc. when possible
- Computationally sane representations for direct/wedge/tensor products.
- Any linear algebra-ish operation currently available on commercial software.

Nice Rings

When I say "nice" rings, I'm thinking of:

- $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{Z}/n\mathbb{Z}$, finite fields
- Polynomial rings over $\mathbb{Z}$, $\mathbb{Z}/n\mathbb{Z}$, finite fields
- Dedekind domains and their fraction fields
- Local Rings, Local Fields

Parallel Operation

Parallel operation on my desktop:

- Should make reasonable use of multi-core stand-alone systems.
- Should benifit from multi-core even when tasks aren’t ridiculously time intensive.

Parallel Operation

Parallel operation distributed-memory systems:

- Shift from shared to distributed memory should be transparent.
- Should scale to clusters and supercomputers with nothing more than a re-compile.

Drop-in Functionality

- Should easily replace currently linear algebra in packages such as Pari and Macaulay 2.
- Should be easily integrated into emerging packages (e.g. FLINT, Sage)

Portability

- Pure ISO C (C++, Python wrappers for usability)
- Dependence only on well established, well supported libraries
- 64 bit clean, byte-order neutral
- Thread safe (all functions re-entrant)
- Simple configuration and build.

Maintainability

- Good version control
- Painfully rigorous coding standards.
- Documentation for all functions generated from code comments.
- Boundary check and random feed tests included for all library functions.
- Pool of dedicated programmers.

Things I’m considering

- Runtime profiling.
- Runtime tuning.
- Sparse Object / Black Box support.
- Fault tolerance.

Design decisions so far

Modular (as in programming) design with recursive data types.

- Object oriented data-method association (yes, you can do it without C++)
- Generalize algebraic routines to ring operations.
- Sacrifice some speed for specific base ring cases for greater flexibility.
- Leave hooks for optimized version of routines for special cases for those who want them.

Design decisions so far

Use pthreads/MPI hybrid approach?

- Use MPI for distributed memory communications, pthreads for shared memory.
- Personal benchmarking for single machine shows pthreads significantly faster than MPI for some naive tests.
- Allows single machine users to use library even if they don’t have MPI.
- Might be a mistake! Perhaps it’s best to just use MPI. What does the audience think?

Design decisions so far

Realistic Development Approach

- First deal with $\mathbb{Z}$.
- Initial development will focus on getting coding infrastructure in place.
- Goal for API specification is May 2007.
- Initial release goal for “working” code is August 2007.

What I need from you

- What features do you need?
- What should the API include?
- Implementation advice?
- Volunteers??