SDPB:
Bigger Models and Faster Solutions

Walter Landry

Caltech

What is SDPB?

- Open source, arbitrary precision, parallelized semidefinite program solver, designed for the conformal bootstrap.
- Builds on Linux, Mac, and Windows.
- Python wrappers are available.

SDPB Structure

- From a computer science perspective, much of what SDPB does is operations on independent blocks of matrices.
- The results of these independent operations are combined into a single, not overly large matrix Q.
- What makes this exercise difficult and slow is the need for extremely high precision arithmetic.

Speeding up SDPB

- SDPB is already parallelized with OpenMP
- This works pretty well, but only scales up to a single hardware node.
- This limits the kind of problems you can work on. Either because they take too long or they require too much memory.

Distributed Parallelism

- The usual solution is to distribute the computation across multiple network-connected nodes.
- Good performance requires restructuring the code. You have to be very conscious of when communication happens.

Parallelism is Complicated

- It gets really complicated really fast.
- Each jump in the number of cores (1 to 10, 10 to 100, 100 to 1000) requires significant effort.
- This is why you generally want to use a library that has already figured many of the issues.

Elemental

- A library oriented towards operations on distributed, dense matrices.
- It has been run on machines with thousands of cores.
- Comes with support for arbitrary precision arithmetic using MPFR.
- I made a fork which uses the faster GMP library.

Distributed SDPB

- I have modified SDPB to use Elemental for linear algebra.
- The operations on the conformal blocks can all run on separate, distributed cores.
- The assignment of blocks to cores is extremely simple (round robin).
- This ***should*** work well enough when there are many more blocks than cores.

Scaling Q Computation

- For my benchmark, this takes about 1/3 of the total run time.
- It involves squaring a long, skinny matrix.
- It is not all that simple to parallelize, but there has been lots of work by many clever people to make linear algebra fast.

Faster?

- On my 4-core laptop.
  - Q computation runs within 5% of the OpenMP version.
  - Overall it is a bit slower (15%).
  - The parts that should scale perfectly do not due to CPU scaling.
- The next step is to run on cluster hardware.

Limits on Parallelism

- The Ising model has 1264 blocks.
- With the right machine, we would like those parts to run 1264 times faster.
- But not all blocks are the same size.

Intelligently Allocating Cores

- However, larger blocks can still benefit from using multiple cores.
- Then you have to figure out how to allocate cores to nodes.
- You do not want to split a block amongst cores that belong to different nodes.

Bigger is Better

- For 1x32 and 2x32 cores, ratio of time to square a matrix
  - 9224 x 368 elements: 1.15
  - 36896 x 1536 elements: 1.94
  - 2.0 is perfect scaling

Other Approaches

- It is odd that we have to use such extreme precision to get useful answers.
- Is there a way to subtract off the infinities so that the matrices are better conditioned?

Scaling Input Files

- Each core reads the entire XML input files into custom data structures in memory.
- These data structures are then converted into GMP numbers and discarded.
- For large input files, this ephemeral per-core memory requirement can be larger than what is needed for the actual computation.
- My plan is to change to a different parser (libxml2?) that use incremental parsing.

Open Development

- The code is available on Github and Gitlab

  `https://github.com/davidsd/sdpb/`

    `elemental` branch

  `https://gitlab.com/bootstrapcollaboration/elemental`

    `no_warnings` branch

- Discussion is on a public mailing list

  `https://groups.google.com/forum/#!forum/bootstrap-collaboration-software`