Branson

An IMC transport mini-app for studying new parallel algorithms

Description

Branson is a proxy application for the Implicit Monte Carlo method. IMC models the exchange of radiation with material at high temperatures. Implicit Monte Carlo is known to be memory latency bound, suffer from load balance issues and require a large amount of parallel communication.

Source

Two source files for Branson are available. One uses ParMetis and the other uses Metis.

ParMetis version: branson-xroads-v1.0.0.tgz on Crossroads website (tag 0.81)

Metis version: branston-0.82.tar.gz on Crossroads website or clone from github https://github.com/lanl/branson using commit 8b42c3caa37d563a27a85519522f236dfd8fd159 (tag 0.82)

The Trinity FOM was run using the ParMetis version of Branson (tag 0.81). Submissions may use either version of Branson described above to calculate the FOM and should specify the version used in the proposal.

Mechanics of Building

- Clone Branson from github: https://github.com/lanl/branson or untar the source.
- Branson requires MPI 3.0+, Boost (headers only), Metis and ParMetis. The module system at Los Alamos correctly sets environment variables for these packages that are picked up by CMake.
- Build with CMake, the CMakeLists.txt is in `branson/src/`. There are two Branson specific options at the configure stage (in addition to the standard CMake options):
  - cmake -DCMAKE_BUILD_TYPE=<Debug|Release> -DN_GROUPS=<[0-9]+> /path/to/branson/src/
- For the benchmark problems Branson should be built with 50 groups
- If the module system does not set environment variables correctly for package dependencies they will need to be changed. The environment variables for each package are in the CMakeLists.txt file in `branson/src/`
- Compile branson with "make" in the build directory (the directory where the CMake was called from)

Mechanics of Running

- Branson is MPI only and should be run with srun, aprun, etc. The syntax is:
  - srun -n <[0-9]+> --cpu_bind=cores ./BRANSON /path/to/input
  - You may have to substitute a different command for "--cpu_bind=cores". This option does not significantly impact runtime as long as the MPI distribution places memory sensibly.

Baseline Run

- The input files for the baseline runs are found in /branson/inputs/. The filenames are “proxy_small.xml”, “proxy_med.xml” and “proxy_large.xml” corresponding to the small, medium and large runs.
- If you need to change the amount of memory used in a problem you can adjust the number of particles by changing the “photons” field in the input file. This is the global number of particles, across all processors. Each particle is 88 bytes and the load balancing routines will evenly distribute them across processors such that the particle memory use per core is roughly (n particles/n ranks per node)*88 bytes/particle.
- In the baseline runs the other parameters should stay the same (do not change the timestep size, opacity, region layout, heat capacity, temperature or number of cells)

Optimized Run

- There are a few parameters in the input files you can change which relate to parallel run modes:
    - grip_size: The desired number of cell to be retrieved together, think of this as a prefetch size for off processor mesh data. It also serves to balance MPI latency with bandwidth. Recommended value is 10000.
    - batch_size: The number of particles to process between processing parallel messages. There is some amount of work that requires off processor data, this lets you balance checking for completed messages and doing available work. Recommended value is 5000.
    - dd_transport_type: This is the method for getting off rank mesh data. The "CELL_PASS" option uses two-sided MPI semantics---ranks ask remote ranks for mesh data with send cell IDs/receive cell IDs/send cells/receive cells patterns. "CELL_PASS_RMA" uses one-sided MPI semantics---ranks read the remote data they need directly without involving the remote rank. Recommended value is "CELL_PASS" unless specific hardware support for one-sided RMA operations exists (e.g. RDMA).
- Due to limitations in ParMetis (it runs out of memory when more than a billion cells are used), the large proxy run requires a simple cube decomposition and thus requires a number of ranks with an integer cube root.

Correctness

- The output of Branson can be assumed to be correct if the values for radiation conservation and material conservation are on the order of 1.0e-13 relative to the emission energy and census energy.

    ```
    $ grep "conservation" output_proxy_small.txt
    output_proxy_small.txt:Radiation conservation: 6.94567e-19
    output_proxy_small.txt:Material conservation: -1.02999e-18

    $ grep "Emission E": output_proxy_small.txt
    ```

output_proxy_small.txt:Emission E: 1.55491e-05, Absorption E: 2.31953e-05, Exit E: 2.99596e-15

Figure of Merit

- The figure of merit is the total transport time, which is printed to standard out after the simulation is completed (it does not include mesh setup or reading the input file). The time for the large problem will be used for the Crossroads Benchmark.

  $ grep "Total transport" output_proxy_small.txt
  Total transport: 51.9266

Benchmark results

Benchmark timings were provided using Trinity Haswell nodes with 32 ranks/node.

Small problem (proxy_small.xml, 1 node, 32 cores) – 1052.88 seconds

Medium problem (proxy_med.xml, 64 nodes, 2048 cores)—570.68 seconds

Large problem (proxy_large.xml, 3456 nodes, 110592 cores) –393.55 seconds

It should be noted that the large problem does not use the full 400 TB. For the large run, the problem is sized to be about 1/8 of the node's memory, which is about 50 TB of memory across 3456 Trinity Haswell nodes. The size of the problem is determined by the number of cells and the number of particles (photons). Each cell is 1 kilobyte and each particle is 88 bytes. There is some additional data allocated for buffers, but this memory is not expected to exceed 10% of the total particle and mesh memory.

The **total memory per node** can be calculated by:

Particle Memory: (n particles/number of nodes) * 88 bytes/particle
Mesh Memory: (n mesh cells/number of nodes) * 1000 bytes/cell
Buffer Memory: (12 * 6 * grip size + map size)*n ranks per node *1000 bytes/cell

Note: Most of the time the buffer memory is small compared to the mesh and particle memory. There is also some dynamic memory growth in the waiting list (particles that are waiting for mesh data to transport) which cannot be calculated directory but is generally about 10% of the particle memory.

Reporting

For the electronic submission, include all the source and the makefiles used to build on the target platform, input files and runscripts. Include all standard output files.

Runs should use between 1/8 and 1/4 of the available on node memory using the total memory per node. The number of photons can be changed per the run rules but not the mesh cells. Report the amount of memory used per node for final submission and associated input file.

Crossroads Change Log

8/20/2018 - Updated Branson source code to fix a bug that caused a hang when the number of processors was not a multiple of 8

9/11/2018 - Updated Branson input file proxy_large.xml, line 48 to set y_end=1.2 (not 1.0) for uniform spacing.

1/28/2018 - The release is tagged on github as “Branson 0.81”. Here is a change log from the last release for crossroads procurement:

Important changes

- **decompose_mesh.h:** Use a simpler, less memory intensive index exchange in mesh remapping
- **rma_tally_manager.h:** Add assertions for RMA tally operations to assert that the window will always have the same displacement and that there are no conflicting locks on the window
- **mesh_pass_transport.h:** Statically size the census list during transport, "pushing back" was the main cause of memory growth beyond what was expected.
- **mesh_request_manager.h:** Statically size the receive and send buffers for exchanging mesh data
- **mesh_request_manager.h:** Use const references when returning received cell lists and calculating census energy (reduces memory use).
- **input.h:** Use PUGI XML to parse XML input files instead of Boost XML reader
- **input.h:** Only read the input XML file with the root rank and broadcast the data to all other ranks
- Add PUGI XML to the source directory (this removes the Boost header requirement)
- Use a slightly smaller grip size in **proxy_small.xml** to avoid running out of memory on trinitite nodes

Minor Changes

- Remove Boost checks in the CMake system
- Build PUGIXML with CMake system
- Update the CMake build system to use more modern CMake (work done by Kelly Thompson)
- Use "Proto_Mesh" and "Proto_Cell" classes to do mesh decomposition and remapping with a lighter weight cell class. "Proto_Cell" does not have opacity, temperature or heat capacity and has about 80% less memory for the 50 group case
- Split off Proto_Mesh functions from standard Mesh functions.
- Clang-format all files in the src/ directory
- Add continuous integration system