Kokkos – A Case Study

Daniel Holladay

July 2, 2019

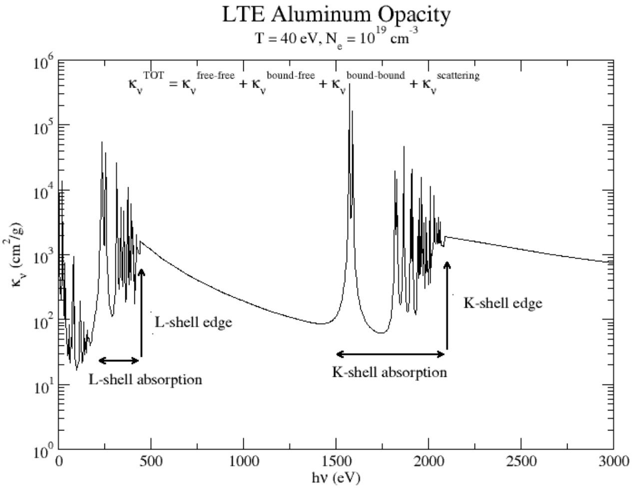
Mentors: Chris Werner, Chris Fontes, Todd Urbatsch

LA-UR-19-26019

Kokkos Usage

- View of views
- Team parallelism
- Team scratch memory
- Atomic, RandomAccess, and Unmanaged memory traits
- `parallel_{for,reduce,scan}` patterns
- Custom reduction types
- Dynamic scheduling
- **Soon:** TeamVectorRange

Opacity: Pictographic Representation



Credit: Chris Fontes – XCP-5

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\left(\mathbf{r}\right), T_e\left(\mathbf{r}\right), T_r\left(\mathbf{r}\right), h\nu\right) =$$

$$\sum_{i\ell jm} \frac{N_{i\ell}\left(\rho\left(\mathbf{r}\right), T_e\left(\mathbf{r}\right), T_r\left(\mathbf{r}\right)\right)}{\rho\left(\mathbf{r}\right)} \left(\sigma^{\text{(bound–bound)}}_{i,\ell\rightarrow m}(h\nu) + \sigma^{\text{(bound–free)}}_{i\ell\rightarrow jm}(h\nu)\right)$$

$$+ \frac{N_e\left(\mathbf{r}\right)}{\rho\left(\mathbf{r}\right)} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{\text{(free-free)}}(h\nu)$$

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\,(\mathbf{r}), T_e\,(\mathbf{r}), T_r\,(\mathbf{r}), h\nu\right) =$$

$$\sum_{i\ell jm} \frac{N_{i\ell}\left(\rho\,(\mathbf{r}), T_e\,(\mathbf{r}), T_r\,(\mathbf{r})\right)}{\rho\,(\mathbf{r})} \left(\sigma_{i,\ell\to m}^{\text{(bound–bound)}}(h\nu) + \sigma_{i\ell\to jm}^{\text{(bound–free)}}(h\nu)\right)$$

$$+ \frac{N_e\,(\mathbf{r})}{\rho\,(\mathbf{r})} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{\text{(free-free)}}(h\nu)$$

League of Teams – Spatial Cells

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\,(\mathbf{r})\,,\,T_e\,(\mathbf{r})\,,\,T_r\,(\mathbf{r})\,,\,h\nu\right) =$$

$$\sum_{i\ell jm} \frac{N_{i\ell}\left(\rho\,(\mathbf{r})\,,\,T_e\,(\mathbf{r})\,,\,T_r\,(\mathbf{r})\,\right)}{\rho\,(\mathbf{r})} \left(\sigma^{\text{(bound–bound)}}_{i,\ell\to m}(h\nu) + \sigma^{\text{(bound–free)}}_{i\ell\to jm}(h\nu)\right)$$

$$+ \frac{N_e\,(\mathbf{r})}{\rho\,(\mathbf{r})} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{\text{(free-free)}}(h\nu)$$

League of Teams – Spatial Cells
Team of Threads – Photon energies

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\,(\mathbf{r}),\, T_e\,(\mathbf{r}),\, T_r\,(\mathbf{r}),\, h\nu\right) =$$

$$\sum_{i\ell j m} \frac{N_{i\ell}\left(\rho\,(\mathbf{r}),\, T_e\,(\mathbf{r}),\, T_r\,(\mathbf{r})\right)}{\rho\,(\mathbf{r})} \left(\sigma^{\text{(bound–bound)}}_{i,\ell\to m}(h\nu) + \sigma^{\text{(bound–free)}}_{i\ell\to jm}(h\nu)\right)$$

$$+ \frac{N_e\,(\mathbf{r})}{\rho\,(\mathbf{r})} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{\text{(free-free)}}(h\nu)$$

League of Teams – Spatial Cells
Team of Threads – Photon energies
Serial – Ion stages

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\,(\mathbf{r}), T_e\,(\mathbf{r}), T_r\,(\mathbf{r}), h\nu\right) =$$

$$\sum_{i\ell jm} \frac{N_{i\ell}\left(\rho\,(\mathbf{r}), T_e\,(\mathbf{r}), T_r\,(\mathbf{r})\right)}{\rho\,(\mathbf{r})} \left(\sigma^{(\text{bound–bound})}_{i,\ell\to m}(h\nu) + \sigma^{(\text{bound–free})}_{i\ell\to jm}(h\nu)\right)$$

$$+ \frac{N_e\,(\mathbf{r})}{\rho\,(\mathbf{r})} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{(\text{free-free})}(h\nu)$$

League of Teams – Spatial Cells
Team of Threads – Photon energies
Serial – Ion stages
Vectors – Atomic Transitions

Calculating an Opacity

The opacity in a material of density ρ, electron temperature T_e, and radiation temperature T_r, is given by:

$$\kappa_{tot}\left(\rho\,(\mathbf{r}),\, T_e\,(\mathbf{r}),\, T_r\,(\mathbf{r}),\, h\nu\right) =$$

$$\sum_{i\ell jm} \frac{N_{i\ell}\left(\rho\,(\mathbf{r}),\, T_e\,(\mathbf{r}),\, T_r\,(\mathbf{r})\right)}{\rho\,(\mathbf{r})} \left(\sigma^{(\text{bound–bound})}_{i,\ell\to m}(h\nu) + \sigma^{(\text{bound–free})}_{i\ell\to jm}(h\nu)\right)$$

$$+ \frac{N_e\,(\mathbf{r})}{\rho\,(\mathbf{r})} \int_{-1}^{1} \sigma_s\left(\mu, h\nu\right) \mathrm{d}\mu + \kappa^{(\text{free-free})}(h\nu)$$

League of Teams – Spatial Cells
Team of Threads – Photon energies
Serial – Ion stages
Vectors – Atomic Transitions
Atomic Populations – calculated prior to opacity calculation but in the same league team loop

Calculating the Atomic Populations

Solving for populations involves constructing and solving a block tri-diagonal linear system many times.

```
parallel_for cells
  calc_populations –
    while not_converged {
      Ne = get_ne_guess()
      build_matrix –
        parallel_for ion_stages
          parallel_for transitions
      solve_matrix –
        serial_for block_rows
          parallel_for matrix_rows
            parallel_for matrix_columns
      calc_residual
    }
  calc_opacity –
    parallel_for photon_energies
      serial_for ion_stages
        parallel_for transitions
```

League of Teams
Team of Threads
Serial
Vectors

Test Problem

- Only one spatial cell: $\rho = 10^{-3}$ g/cc, $T_e = 5$ eV, $T_r = 20$ eV.
- Iron opacity calculated from a reduced atomic model (rDCA) meant for inline implementations like this one.
  - 27 ion stages – number of bound electrons + 1 for case when atom has no bound electrons
  - $\sim$ 28 energy levels per ion stage
  - $\sim$ 750 unknowns.
  - $\sim$ 12k photo-transitions
- Continuous energy total opacity evaluated at 500k logarithmically spaced photon energies between $10^{-2.5}$ eV and $10^{1.5}$ eV.
- On the GPU: implemented photon energy domain decomposition to maximize GPU resource usage.
- On the CPU: all resources allocated to a single thread team.

Performance Results

- Power 9 Node: gcc 7.4.0, cuda 9.2
  - Power9 OpenMP – 5.51 s
  - 1 V100 cuda – 0.65 s
  - GPU speedup – *8.5x*
  - With power consumption – **9.1x**
- CTS-1 (LANL Grizzly) node: Intel 18.0.2
  - OpenMP – 5.76 s
  - GPU speedup – *8.9x or 9.8x*
  - With power consumption – **7.1x or 7.8x**
- X86 V100 Node: Intel 19.0.2, cuda 10.1
  - Skylake OpenMP – 6.29 s
  - 1 V100 cuda – 0.59 s
  - Single GPU speedup – *10.7x*
  - With power consumption – **7.5x**

Takeaways

- We see between 7-9x improvement in performance per Watt for this test case using the V100 over 3 different CPUs
- Kokkos enabled GPU speedup despite most of my effort being focused on optimization for commodity type clusters
- We find the Kokkos hierarchical parallelism:
  - maps well to this application
  - provides tools to concretely express complex parallelism

Demo