LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-699797

Cretin Memory Flow on Sierra

S. H. Langer, H. A. Scott

August 5, 2016

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Cretin Memory Flow on Sierra
Steven Langer and Howard Scott
LLNL
July 21, 2016

Introduction

The Cretin iCOE project has a goal of enabling the efficient generation of Non-LTE opacities for use in radiation-hydrodynamic simulation codes using the Nvidia boards on LLNL's upcoming Sierra system. Achieving the desired level of accuracy for some simulations require the use of a vary large number of atomic configurations (a configuration includes the atomic level for all electrons and how they are coupled together). The NLTE rate matrix needs to be solved separately in each zone. Calculating NLTE opacities can consume more time than all other physics packages used in a simulation.

Atomic database

Information about the atomic configurations and the transitions between them is stored in a "database". The database might be as large as 16 GB. We hope that careful choices for the set of atomic configurations will reduce it down to 8 GB or less. The size of the database should scale somewhat more slowly than quadratically with the number of configurations in the database. The reason is that larger databases have a higher percentage of configurations with more than one excited electron. The database only includes single electron transitions, so the percentage of states that are coupled drops as the fraction of multiply excited configurations increases.

The collision rates in a database are all computed using the same functional form (for current databases). The form can vary between databases. The most popular current form stores 8 numbers per transition.

Photo-excitation rates may involve an integral over the line shape for a transition and the amount of work varies from one transition to the next.

All rates of a given type (e.g. collisional excitation) are adjacent in the database. Rates for different physical processes are stored back-to-back in the database. The simplest approach is to load the entire database into the memory of each GPU board. All transitions have an inverse transition (i.e. going the opposite direction between the two configurations). It is easy to program as if there are separate databases for collisional excitation/de-excitation, collisional ionization/recombination, radiative excitation/de-excitation, radiative ionization/recombination, and auto-ionization/di-electronic recombination.

Transition Rates

Cretin takes the current populations of the configurations and the radiation field in a zone as input. New populations (and some equation of state, EOS, related values) are outputs.

Cretin runs through the entire set of transitions generating a “transition descriptor” for each one. A descriptor has configuration numbers for the initial and final state, a transition rate, and other information. Cretin puts 14 numbers in a descriptor, but HYDRA (the biggest current customer for NLTE opacities) probably only needs 6 or 8 of them.

Collision rates might be computed in groups of 4 so that a warp can fetch 32 coefficients (4 groups of 8). It would also be possible to give each thread in a warp its own collision. As it stands, that would lead to stride 8 accesses. If each thread has 8 registers free, a warp could do 8 aligned fetches and then perform a transpose so that each thread has its 8 coefficients. It would also be possible do the transpose on the Power 9 host. Before transfer to the GPU. That is fine if a warp always starts processing transitions on a “warp-aligned” transition.

Electron densities can be high enough that continuum lowering causes some configurations to disappear (the configuration is no longer bound). Transitions for those configurations will still be present in the transition database. Cretin currently uses an indirection array so that it only processes transitions between bound configurations. That approach minimizes the work, but indirection usually has a negative impact on memory performance.

As an alternative, a thread could use a predicate to disable storage of rates when it notices that it is working on a transition involving a configuration that is not bound at the current density. It might also be possible to compute the rate and ignore it later on. We need to be sure it is safe to calculate rates for an unbound state before trying the later approach.

Photo-excitation rates can be computed at line center (good for optically thin lines), in which case a warp should compute multiple rates at a time. Rates might also be computed using a quadrature (required for strong or optically thick lines), in which case a warp might compute one rate at a time. The number of energies may not be an integral multiple of the number of threads in a warp so zero padding or predication might be required. If the number of photon energies in an integral is small compared to the size of a warp, it might be better to have each thread do the integral for a different transition. That choice would change the optimal layout of data in memory. Having each thread handle a full transition would be more like the planned approach for collisions and would suggest a similar transpose of database values.

Loop Order

The outermost loop currently runs over zones. Nested inside that (in mixed zones) is a loop over elements. The worker loop calls a function that in turn calls a function to compute collision rates.

Given limited amounts of high bandwidth memory, we probably want the outermost loop to be over elements when using an Nvidia board. That allows us to load the transition database for an element to HBM at the start of an element and process all zones before

loading the next database. If we keep the current loop order, the databases for all elements need to fit in HBM or we will need to do a lot more data loading.

Solving the rate equations for a single zone might utilize an entire Nvidia board for a large database. We will also need an option to process several zones simultaneously on a single Nvidia board. A simple database will not have enough transitions to occupy an entire board and some ionization states will have so few active configurations that they can't occupy an entire board even if the database is large. When using a large database there may be cases where only a single zone can fit on a board.

We will either use a separate kernel call for each zone or modify the way Cretin works so that we can batch multiple zones into a single kernel invocation. If we launch multiple kernels at the same time, we will (probably) wait for all of them to finish before launching another kernel. The zones that are launched together must be chosen so as not to overflow the memory of the Nvidia board. Launching new zones as old ones finish would make it more complicated to avoid running out of memory.

Outputs from Rate Evaluation

The result of the rate calculation is a long list of transition descriptors. A transition between a pair of configurations may occur multiple times (e.g. both collisional and radiative excitation). The rates are used to populate a dense matrix that is then solved using LAPACK functions or as inputs to a sparse matrix solver. The rate list is comparable in size to the atomic database. The rate list is populated in "stride one" order (modulo the fine details of when individual warps tack new data onto it).

A Possible Strategy for Kernel Launching

Our goal is for the database, rate list, and final rate matrix to all fit in fast memory. The database is used by many zones and the rate list never needs to be copied back to the host if the rate equations for a zone are completed before moving on to the next zone.

The zonal populations and radiation field will probably remain in DDR and be read directly by the Nvidia board. They should be accessed in stride one order. Their size is small enough that DDR bandwidth should be fast enough that is not a bottleneck.

We would like to move the atomic database to the Nvidia board once at the start of the pass over an element in each time step. If there isn't enough memory to do that, it will be moved at the start of each zone.

The rates will be allocated using in-package (fast) memory. At the end of the rate calculation phase, the rate table will be comparable in size to the atomic database because there is one descriptor for each active transition in the database.

The solver can free the atomic database if memory is needed to make room for the rate matrix. The solver can run using the rates that are already in fast memory. The rates should never need to be stored back to DDR.

The memory access pattern is reasonably hardware friendly. The loop over elements should not be a problem. The loop over zones is stride one in host memory and will probably be fast enough without an explicit copy of the population array to the Nvidia board. The transition rates will probably include a separate kernel launch (at least initially) for each type of transition. The transition rates of a given type are accessed in memory order and the transition descriptor table is written in memory order.

We need to work on different transitions simultaneously on different warps to get enough parallelism. Transition descriptors are of a fixed size, so it should be easy for each warp to figure out where in the list to put its rates.

If transition rates are never copied back to DDR and there is enough fast memory that the atomic database never needs to be "unloaded", the traffic between host and accelerator is just the atomic configuration population and radiation field for each zone. This is small enough to pass in each kernel call.

If the atomic database must be removed before the rate equations are solved, moving it to the Nvidia board at the start of each kernel launch could consume significant time. It can't be pre-staged because space was required for the rate matrix. Squeezing a database down until it fits will be a popular option.

A kernel will compute all transitions with a single call even if there are more transitions than "cores". The Nvidia board will schedule warps as it sees fit. That means there is no guarantee that transition rates will be pasted into the output list in memory order. That shouldn't be an issue because each warp will write something like 8*32 numbers into the output array and Nvidia boards are tolerant of memory latency due to the large number of warps.

Questions that need to be answered soon

1) Do we need to transpose rate coefficients?
2) Can we fit the atomic database, a full list of transitions, and solve the rate matrix without kicking the database out of HBM?
3) Can a single MPI process have kernels for several zones running simultaneously?
4) Is the time for a kernel launch large enough that we will need to batch multiple zones into a single kernel launch?
5) Is it practical for several MPI processes to share an Nvidia board (via time slicing given the constraints on memory footprint)? If not, we will idle some MPI processes and have the remaining processes compute all the rates. Moving work to a subset of processes would be easy to add to the global load balancer that will be written by this project.

This document was released as LLNL-TR-699797. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC.