Michael D. Walker (mw6136)
COS597E: Programming Languages, Distributed Systems
October 9, 2024

# Final Project Proposal: Dynamic Load Balancing in a Massively Parallel Reacting Flow Solver

## 1. Introduction

Modern Computational Fluid Dynamics (CFD) software relies on distributed-memory parallel computer architectures for large-scale computations. Using resources efficiently while at the same time maintaining high accuracy of the numerical solution is a major challenge. *Computational load imbalance* is a well-known performance issue in multiprocessor reacting flow (e.g., combustion) simulations utilizing directly integrated chemical kinetics. **This project proposes to implement a *dynamic load balancing* scheme (specifically Lend When Idle) on my research group's (lab: CTRFL) low Mach number flow solver, `NGA` [1, 2].** Additionally this project proposes to develop a *reference mapping* paradigm to sort group cells with similar thermochemical composition, thereby evaluating their chemical source terms only when sufficiently dissimilar.

## 2. Background

`NGA` is a 3-D finite-difference energy-conserving incompressible flow solver, capable of solving the variable-density Navier-Stokes equations on structured Cartesian meshes using a fractional-step method. It was written for massively parallel large-eddy simulation (LES) and direct numerical simulation (DNS) of both premixed and non-premixed reacting turbulent flows. It is parallelized using a hybrid `MPI`/`OpenMP` approach and is highly scalable. `NGA` uses an operator splitting technique to decouple the conservation equations and chemical source term calculations. With this technique, chemistry can be treated as an independent stiff ordinary differential equation (ODE) system within each computational cell. `NGA` then uses `SUNDIALS` [3] (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) `CVODE` (unsteady solver) for solving stiff systems of initial value ODEs for detailed chemistry (chemical Jacobian construction, inversion, and Newton iteration).

Typically, the computational cost of the chemical source term evaluation dominates the performance metrics and creates an un-even computational load distribution in parallel applications. This is discussed in more detail in Appendix A, and furthermore chemical Jacobian evaluation and factorization is the most expensive step in combustion simulations. Load balancing is not handled internally by the `SUNDIALS` integrators but rather is the responsibility of the application developer. As a result of the highly non-linear characteristics of chemical kinetics, a large variation in the convergence rates of the ODE integrator may occur, leading to a high load imbalance across multiprocessor configurations. However, the independent nature of chemistry ODE systems leads to a problem that can be parallelized easily (embarrassingly parallel) during the flow solution. The presented model takes advantage of this feature and balances the chemistry load across available resources.

## 3. Systems Component: Dynamic Load Balancing

Lend When Idle [4, 5, 6] (LeWI) is a novel *work stealing* [7] algorithm that distributes resources equally among `MPI` processes in a node while they are doing computation, and re-assigns resources of `MPI` processes while they are blocked in communication calls. One of its main properties is that the load balancing is task neutral, done at runtime without analyzing nor modifying the application previously.

This will be implemented for the `NGA` combustion subroutine `combustion.f90`, which is the most poorly balanced. For the reasons discussed in Appendix A, chemical Jacobian construction, factorization, and inversion occurs in this subroutine, so it is also the most costly. LeWI will then be implemented in as many other subroutines as time permits (e.g., boundary condition enforcement `borders.f90` and mesh discretization `geometry.f90`).

## 4. **Programming Languages Component: Reference Mapping**

The timestep value and hence the total number of floating-point operations during the integration depends on the initial thermochemical composition. Parallelization is achieved with geometrical domain decomposition, leading to explicit chemistry load imbalance due to spatially and temporally varying values. A simple reference mapping feature will allow a further reduction in computational cost. This approach groups cells sharing similar thermochemical composition values together and solving the chemistry only once for this group [8]. Such a mapping approach is intended to be used for regions with low reactivity, (e.g., where no fuel is present or in low-temperature regions far upstream and downstream of the flame, where few chemical reactions are occurring). **The reference mapping acts as a filter for load balancing, where the reaction rates of cells satisfying a user-given criteria are copied from a reference cell solution.** At a given time instance, a reference cell is picked and the chemistry source term of that cell is solved and copied to other reference cells. The criteria used for identifying the reference cells is: $Z_i < Z_{\text{tol}}$, $|T_i - T_{\text{ref}}| < T_{\text{tol}}$, where $Z_i$ and $T_i$ are mixture fraction (the mass fraction of fuel in a fuel/oxidizer stream) and temperature of $i$-th cell, respectively, $T_{\text{ref}}$ denotes the temperature of the chosen reference cell and $Z_{\text{tol}}$ and $Z_{\text{tol}}$ are the user-defined tolerance values, specified in the `input` and `config` files.

Implementing this approach requires additional pairing between `NGA` combustion subroutine `combustion.f90` and mesh grouping subroutine `geometry.f90`. As long as properly strict tolerances are used (chemical reactions are highly non-linear based in an Arrhenius law), the introduced error should be rather small and not affect the global characteristics of the reactive simulation. The reference mapping model should improve the balancing performance modestly by further reducing the load of the more idle processes and increasing their potential to receive more load from the busier processes.

## 5. **Performance Evaluation**

`NGA` currently uses a rudimentary load balancing scheme by approximating the work for each cell ahead of time by tracking the work required to integrate the previous timestep. Thus, it is in the earliest timesteps that more robust load balancing scheme could provide considerable speedup.

Several test cases will be used to evaluate performance. First, to verify proper code execution a fully-developed turbulent pipe flow configuration (presented in Appendix C) will be simulated. This case has previously been used to conduct weak scaling studies of `NGA` with increasing grid resolution. More detailed performance benchmark test cases would be simulated using Large Eddy Simulation data of a hydrogen jet flame [9, 10, 11] and Sandia Flame D [12]. Such simulations were published[13] using `NGA` without load balancing (note the massive time per timestep for the early timesteps of Figures 4 and 5 therein).

## 6. **Proposed Timeline**

Project Milestones:

- By 22OCT (Survey deadline), complete literature review, explore existing load balancing libraries. Gain better understanding of `geometry.f90` subroutine in `NGA` to implement reference mapping.
- By 12OCT (Progress Report deadline), implement LeWI on `combustion.f90`. Demonstrate code execution on annulus pipe flow test case.
- By 25NOV, implement reference mapping with proper modifications to `input` and `config`. Plan to have large jobs ready to submit to `TIGER` and `PERLMUTTER` clusters over Thanksgiving break. Ideally the jet flame benchmark tests, if not, then an annulus pipe flow scaling study up to $2^{14}$ cores.
- Analysis and figure generation until class presentation.
- By 05DEC, complete project presentation. Continue detailed benchmark tests if not completed.
- By 11DEC, complete Final Report.

Timeline mitigations to ensure a deliverable project by the end of the course:

- Implement LeWI only on the `combustion.f90` subroutine, rather than all core subroutines of `NGA`.
- Simulate only the Annulus Pipe Flow test case and conduct an associated scaling analysis, rather than simulating the more complex benchmark cases.

## 7. Personal Benefits

This project would directly benefit my academic research and those of my collaborators (the Computational Turbulent Reacting Flow Laboratory, CTRFL), by potentially considerably improving the performance of our group's primary flow solver. There is ongoing work utilizing In-Situ Adaptive Tabulation (ISAT) [13] to improve memory allocation and retrieval. The proposed project would integrate nicely with that effort to potentially generate publishable results.

## References

[1] O. Desjardins, G. Blanquart, G. Balarac, and H. Pitsch. High order conservative finite difference scheme for variable density low mach number turbulent flows. *Journal of Computational Physics*, 227(15):7125–7159, 2008.

[2] J. F. MacArt and M. E. Mueller. Semi-implicit iterative methods for low mach number turbulent reacting flows: Operator splitting versus approximate factorization. *Journal of Computational Physics*, 326:569–595, 2016.

[3] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.*, 31(3):363–396, 2005.

[4] M. Garcia, J. Corbalan, and J. Labarta. LeWI: A runtime balancing algorithm for nested parallelism. In *Parallel Processing, 2009. ICPP '09. International Conference on*, pages 526–533, Sep 2009.

[5] M. Garcia, J. Labarta, and J. Corbalan. Hints to improve automatic load balancing with lewi for hybrid applications. *Journal of Parallel and Distributed Computing*, 74(9):2781–2794, 2014.

[6] M. Garcia, J. Corbalan, R.M. Badia, and J. Labarta. A dynamic load balancing approach with SMPSuperscalar and MPI. In R. Keller, D. Kramer, and J.-P. Weiss, editors, *Facing the Multicore - Challenge II*, volume 7174 of *Lecture Notes in Computer Science*, pages 10–23. Springer Berlin Heidelberg, 2012.

[7] R. D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, Sep 1999.

[8] B. Tekgül, P. Peltonen, H. Kahila, O. Kaario, and V. Vuorinen. DLBFoam: An open-source dynamic load balancing model for fast reacting flow simulations in OpenFOAM. *Computer Physics Communications*, 267:108073, 2021.

[9] R. S. Barlow. Sandia H2/He flame data. https://tnfworkshop.org/data-archives/simplejet/, 2003. Release 2.0.

[10] R. S. Barlow and C. D. Carter. Raman/Rayleigh/LIF measurements of nitric oxide formation in turbulent hydrogen jet flames. *Combustion and Flame*, 97(3):261–280, 1994.

[11] R. S. Barlow and C. D. Carter. Relationships among nitric oxide, temperature, and mixture fraction in hydrogen jet flames. *Combustion and Flame*, 104(3):288–299, 1996.

[12] R. S. Barlow and J. H. Frank. Effects of turbulence on species mass fractions in methane/air jet flames. *Symposium (International) on Combustion*, 27(1):1087–1095, 1998. Twenty-Seventh Sysposium (International) on Combustion Volume One.

[13] C. E. Lacey, A. G. Novoselov, and M. E. Mueller. In-Situ Adaptive Manifolds: Enabling computationally efficient simulations of complex turbulent reacting flows. *Proceedings of the Combustion Institute*, 38(2):2673–2680, 2021.

[14] I. Morev, B. Tekgül, M. Gadalla, A. Shahanaghi, J. Kannan, S. Karimkashi, O. Kaario, and V. Vuorinen. Fast reactive flow simulations using analytical Jacobian and dynamic load balancing in OpenFOAM. *Physics of Fluids*, 34(2):021801, Feb 2022.

[15] W. Zhang, A. Myers, K. Gott, A. Almgren, and J. Bell. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications*, 35(6):508–526, 2021.

[16] M. T. Henry de Frahan, J. S. Rood, M. S. Day, S. Hariswaran, S. Yellapantula, B. A. Perry, R. W. Grout, A. Almgren, W. Zhang, J. B. Bell, and J. H. Chen. PeleC: An adaptive mesh refinement solver for compressible reacting flows. *The International Journal of High Performance Computing Applications*, 37(2):115–131, 2023.

[17] L. D. Owen, W. Ge, M. Rieth, M. Arienti, L. Esclapez, B. S. Soriano, M. E. Mueller, M. Day, R. Sankaran, and J. H. Chen. PeleMP: The Multiphysics Solver for the Combustion Pele Adaptive Mesh Refinement Code Suite. *Journal of Fluids Engineering*, 146(4):041103, Feb 2024.

[18] TIGER architecture. https://researchcomputing.princeton.edu/systems/tiger.

[19] PERLMUTTER architecture. https://docs.nersc.gov/systems/perlmutter/architecture/.

## Appendix A. Numerical Scheme and Load Imbalance in Reacting Flow Solvers

NGA [1, 2] is a 3-D finite-difference energy-conserving low-Mach number flow solver, using derivative operators of arbitrarily high order. It is capable of solving the variable-density Navier-Stokes equations on structured meshes using a fractional-step method. Spatial discretization errors are reduced using local $(n-1)$-th order Lagrange polynomial interpolation to achieve $n$-order accurate viscous terms. The code utilizes detailed temperature- and composition-dependent transport and thermodynamic properties, detailed chemical kinetics, and an ideal gas equation of state. NGA was written for massively parallel large-eddy simulation (LES) and direct numerical simulation (DNS) of both premixed and nonpremixed reacting turbulent flows. It is parallelized using a hybrid MPI/OpenMP approach and is highly scalable. NGA uses an operator splitting technique to decouple the conservation equations and chemical source term calculations.

A linear or non-linear system can be defined in matrix form as $\Phi \equiv \varphi_{i,j}$ and $\mathbf{f}(\Phi) \equiv f_{i,j}(\Phi)$ for $i = 1, \ldots, N$, $j = 1, \ldots, N$. Parallelization is achieved with geometrical domain decomposition. The Strang splitting scheme is achieved with the spatially discretized governing equations:

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \underbrace{\mathbf{T}(\Phi)}_{\text{transport}} + \underbrace{\mathbf{S}(\Phi)}_{\text{chemistry}}$$

The time derivative is approximated by finite difference involving the current and future values. The gradient and divergence operators are approximated with linear functions involving multiple neighboring cells. Transport is relatively linear and has a highly structured Jacobian, with weak coupling between species. Chemistry is highly nonlinear and typically solved implicitly. Chemistry terms are local, so each mesh volume can be solved independently (i.e., in parallel). The chemistry and transport substeps are then:

$$\begin{aligned}
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \mathbf{S}(\Phi^{(1)}), & \Phi^{(1)}(x,0) &= \Phi(x,t_n) & &\text{on } [t_n, t_n + \Delta t/2] \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \mathbf{T}(\Phi^{(2)}), & \Phi^{(2)}(x,0) &= \Phi^{(1)}(x,\Delta t/2) & &\text{on } [t_n, t_n + \Delta t] \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \mathbf{S}(\Phi^{(3)}), & \Phi^{(3)}(x,0) &= \Phi^{(2)}(x,t_n) & &\text{on } [t_n + \Delta t/2, t_n + \Delta t]
\end{aligned}$$

Considering the time integration of a stiff reacting system, each implicit integration step involves evaluation of the source terms by Jacobian evaluation and factorization in a Newton-Raphson scheme. A system of ODEs in chemistry is defined based on multiple chemical species $N$ (i.e., types of molecules) evolving through coupled reactions $\frac{d\Phi}{dt} = \mathbf{S}\Phi$, where $\Phi = [\varphi_1, \varphi_2, \ldots \varphi_N] \in \mathbb{R}^N$ and $\mathbf{S} \in \mathbb{R}^{N \times N}$. Any time integration scheme requires the eigenvalues of $\mathbf{S}$ to lie inside the stability region of the method. The system is stiff if for the set of the eigenvalues $[\lambda_1, \lambda_2, \ldots \lambda_N]$ of matrix $\mathbf{S}$: $|\lambda_{\max}|/|\lambda_{\min}| \gg 1$ A stiff problem requires a very small timestep for stability with respect to $|\lambda_{\max}|$ in conjunction with a very long time integration to see any significant change in magnitude associated to $|\lambda_{\min}|$.

The system $\mathbf{S}(\Phi) = 0$ is solved first by constructing the Jacobian in 2-D $\mathbb{J} \equiv \partial S_{i,j} \,/\, \partial \varphi_{i,j}$. Then the Newton-Raphson scheme computes the components of the matrix. Each iteration is defined by the linear system $\mathbb{J}(\Phi^{n+1} - \Phi^n) = $ -$\mathbf{S}(\Phi^n)$ and can then be solved through direct matrix inversion sequentially,

$$\Phi^{n+1} = \Phi^n - [\mathbb{J}^{-1} \cdot \mathbf{S}(\Phi^n)] \quad .$$

Here the Jacobian is an $N^2 \times N^2$ matrix and thus **Jacobian evaluation and factorization is the most expensive step in combustion simulations**.

With this technique, chemistry can be treated as an independent stiff ODE system within each computational cell. NGA then uses SUNDIALS [3] (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) CVODE (unsteady solver) for solving stiff systems of initial value ODEs for detailed chemistry (chemical Jacobian construction, inversion, and Newton iteration).

Typically, the computational cost of the chemical source term evaluation dominates the performance metrics and creates an un-even computational load distribution in parallel applications. The difficulties occur due to the intrinsic and non-linear nature of the ODE system. Cost of solving the associated stiff ODEs scales quadratically with the number of species [8, 14]. Furthermore, due to the vast scale separation between the fastest and slowest chemical reaction time scales, the system of ODEs is practically always numerically stiff, requiring the use of implicit time integrators with low timestep values.

Load balancing is not handled internally by the `SUNDIALS` integrators but rather is the responsibility of the application developer. As a result of the highly non-linear characteristics of chemical kinetics, a large variation in the convergence rates of the ODE integrator may occur, leading to a high load imbalance across multiprocessor configurations. However, the independent nature of chemistry ODE systems leads to a problem that can be parallelized easily (embarrassingly parallel) during the flow solution. The presented LeWI model takes advantage of this feature and balances the chemistry load across available resources.

Because the ODEs are uncoupled across the domain, there is considerable freedom in staging their integration. In multi-threaded CPU-based implementations, the total work to integrate all the cells in the domain can be distributed arbitrarily across the available threads with no race or synchronization concerns. The work for each cell can be approximated ahead of time by tracking the work required to integrate the previous timestep. Thus, it is in the earliest timesteps that a Lend When Idle scheme could provide considerable speedup. (Note that unlike the examples [15, 16, 17] presented in the `SUNDIALS` paper, `NGA` does not have adaptive mesh refinement, so it should be a simpler implementation.)

## Appendix B. **Computational Resources**

Computational facilities managed by Princeton University and the Department of Energy (DOE) will be used as part of the proposed project. These computational resources are adequate for moderate-scale Direct Numerical Simulation (DNS) and Large Eddy Simulation calculations at lower Reynolds numbers, and all data post-processing. For larger-scale DNS calculations at higher Reynolds numbers, external computational resources are available through the DOE Computational Science Graduate Fellowship allocation.

`TIGER` [18] is a HPE Apollo cluster comprised of 408 Intel Skylake CPU nodes (2 sockets per node). There are 16,320 processors available (40 per node). Each node contains at least 192 GB of memory (4.8 GB per core). There are also 40 nodes with memory of 768 GB (19 GB per core). Every 40-core node is interconnected (24 nodes per chassis) by a Omnipath fabric with oversubscription. Managed by the Princeton Institute for Computational Science and Engineering (PICSciE).

`PERLMUTTER` [19] is a HPE Cray EX (Shasta) platform, a heterogeneous system of 3,072 CPU-only and 1,792 GPU-accelerated nodes. On the CPU partition, each node each contains two AMD EPYC 7763 (Milan) processors. Every 64-core node has 8 GB of memory per core and is interconnected by one HPE Slingshot 11 NIC (200G + 25 GB/s bandwidth with PCIe 4.0 NIC-CPU connection). The CPU partition achieves performance of 7.7 PFLOPS aggregate peak FP64 and 1536 TB aggregate memory. Managed by the Lawrence Berkeley National Laboratory National Energy Research Scientific Computing Center (NERSC).

## Appendix C. **Scaling of NGA and Benchmark Problem: Annulus Pipe Flow**

A weak scaling study of `NGA` can be conducted using the following simulation of fully developed flow in a pipe and surrounding annulus. The fluid properties correspond to methane in the pipe and air in the annulus. An initial velocity profile is specified, but over time the flow transitions to turbulent and becomes fully developed because a periodic boundary condition is used in the axial $x$ direction.

A scaling study can be conducted by increasing the grid resolution and number of processors while maintaining $\sim 256$ cells per core. Also below is a sample `slurm` script for submitting the job to the scheduler on `TIGER`. It is configured to run the job on 80 processors (2 nodes, 40 cores each). In the `NGA` input file, it is specified that the domain will be split into 8 segments in the y (radial) direction, and each segment will be computed on four processors using `OpenMP`.

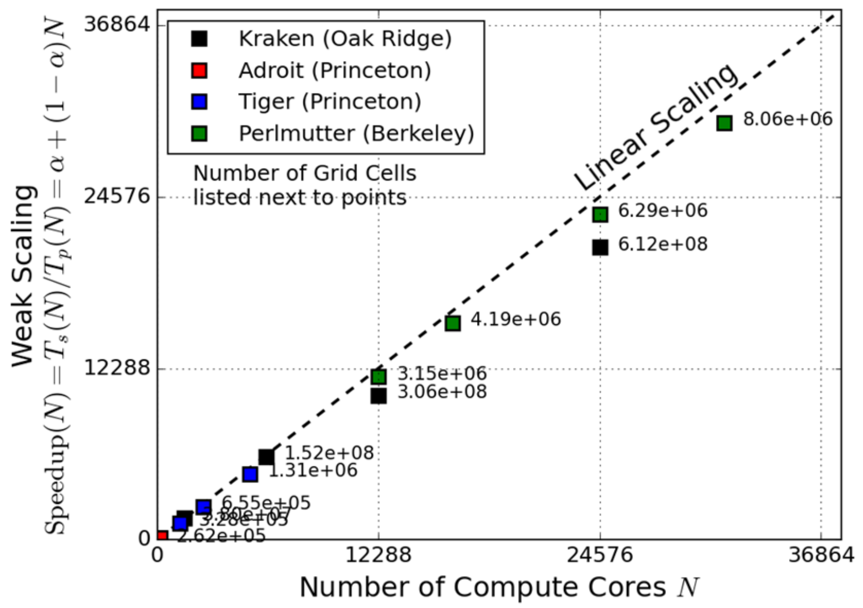

FIGURE 1. NGA exhibits excellent scale-up characteristics. The largest run so far consisted of over 1.6 billion cells, and almost 50,000 compute cores.

```
[language=Fortran]
# INITIALIZATION PARAMETERS ########################
! Name
Simulation :  pipe
! Parameters
nx :   64
nz :   64
Length :  0.04
! Pipes definition
ny :   64 64
Radius :  0.0 0.00225
   0.002 0.00375
Stretching :  2.0 2.0
! Initial velocity field
Mean U Velocity : 67.0 59.5
Mean W Velocity : 0 0
Laminar initial flow : 0 0
Fluctuation rel. amp. : 0.2 0.2
! Files
Init config file : config
Init data file : data.0
# RUNNING PARAMETERS ###############################
! Files
Configuration file :    config
Data file to read :     data.0
Data file to write :    data.1
Data frequency :        1e-4
Optional data to read : optdata.0
Optional data to write: optdata.1
! Inflow file to write : inflow.mz.v57
! Inflow frequency :      0.1e-6
! Inflow location :       0.04
! Notes
! 0 -> 1 : running simulation on 32 processors
! Partitioning
Processors along X :    5
Processors along Y :    4
Processors along Z :    1
OpenMP threads : 4
! Properties            methane         air @ 300K
Chemistry model :       none
Density :  0.651635 1.17214
Viscosity :  1.14597e-5 1.86344e-5
Diffusivity :           0  0
Temperature :           0  0
! Subgrid Scale model
Use SGS model :         .true.
SGS averaging :         Lagrangian
SGS Override Lag. :     .true.
! Body forces
Gravity :  -9.8067 0.0 0.0
! Time advancement
Timestep size :         0.5e-6
CFL number :            2
Subiterations :         2
! End of simulation
Maximum iterations :    5000
Maximum time :          100000
Maximum wall time :     1.0
! Velocity
Velocity conv scheme : 2
Velocity visc scheme : 2
Implicit directions : xyz
! Pressure
Pressure solver : bicgstab
Pressure fft :  .true.
Pressure precond : tridiag
Pressure cvg :  1.E-7
Pressure iterations : 1
! Output
Output type :           ensight-3D
Output frequency :      2e-5
! Statistics
Statistics type : 1D-y
Statistics locations x: all
Statistics frequency : 5e-5
```

```
[language=Fortran]
#!/bin/bash
# HEADER for Parallel job using 80 processors:
#SBATCH --nodes=2               # number of nodes
#SBATCH --ntasks-per-node=10    # number of processors per node
#SBATCH --cpus-per-task=4       # number of cpus per task
#SBATCH -t 1:00:00              # run for 1 hr max
#SBATCH --mail-type=begin       # send email when process begins...
#SBATCH --mail-type=end         # ...and when it ends...
#SBATCH --mail-type=fail        # ...or when it fails.
#SBATCH --mail-user=your_email@princeton.edu    # send notifications to this email
#SBATCH -e job.err              # Name of output file for error messages
#SBATCH -o job.out              # Name of output file for standard output

# BODY - commands to be run
# Load required modules
module load intel
module load intel-mpi

# Set number of openmp threads and number of MPI tasks
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export NTASKS=$(echo "$SLURM_NNODES*$(echo $SLURM_TASKS_PER_NODE | cut -d '(' -f 1)" | bc -l)

echo $OMP_NUM_THREADS
echo $NTASKS

# Print some information
echo Directory is `pwd`
echo Time is `date`
echo
echo This job runs on the following processors:
echo $SLURM_JOB_NODELIST
echo This job has allocated $SLURM_NNODES nodes with $SLURM_TASKS_PER_NODE cores per node.
echo

# run arts
srun -n $NTASKS ~/NGA/bin/arts input_pipe
```