The Relentless Execution Model for Task-uncoordinated Parallel Computation in Distributed Memory Environments

Lucas A. Wilson
Department of Computer Science
The University of Texas at San Antonio
San Antonio, Texas, U.S.A.

Motivation and Problem Statement
Modern High Performance Computing (HPC) systems consist of many thousands of individual servers. As these systems become larger and more complex it will become increasingly difficult, if not impossible, for sufficient numbers of nodes to remain available during parallel computations. Existing execution and programming models are not well suited to future distributed memory parallel systems where hardware reliability cannot be guaranteed.

The Relentless Execution Model (REM) is a dataflow-inspired execution model which relies on uncoordinated parallel processes that interact with a distributed, eventually-consistent key-value store (dictionary), where the values are the data operated on, and the keys inform the runtime task scheduler which operations should be performed on that data.

Contributions
- A theoretical model for the execution of parallel algorithms in a task-uncoordinated fashion, which is capable of executing tightly-coupled, data-dependent algorithms in distributed memory environments where hardware volatility may result in occasional-to-frequent loss of computational processes.
- A domain-specific language, StenSAL, which allows for rapid, easy-to-understand encoding of explicit stencil algorithms targeting the proposed task-uncoordinated model.
- A series of offline optimizations, which can be performed mechanically by the StenSAL compiler, which enable multi-threaded worksharing and vectorization without explicit input from the programmer.
- A mathematical means of determining the minimum amount of memory (and therefore the minimum number of nodes) required to be available in order for a particular stencil algorithm to maintain computational progress.

Components of a Relentless Program
1. a finite set \( T \) of tasks to be performed;
2. a finite set \( L \) of labels, which are used as keys in the shared dictionary;
3. a set \( R \subseteq L \) of result labels whose association with values completes the REM program’s execution;
4. a surjective function \( \text{producer} : L \to T \) that maps dictionary labels to the task that produces the value for that label;
5. a function \( \text{reducer} : T \to P(L) \) that maps each task to the labels of the inputs the task requires before it can be executed; and
6. a function \( \text{comparer} : T \to \{L \to V \to \{L \to V \} \to \{L \to V \}\} \) that maps each task to the partial function it computes.

The Relentless Task Search Algorithm
1. for all \( r \in \text{Result} \) do
2. \( \text{SOLVE}(r) \)
3. end for
4. function \( \text{SOLVE}(\text{label}) \) then
5. task \( \leftarrow \text{PRODUCER}(\text{label}) \)
6. Missing \( \leftarrow \{ l | \text{task} \in \text{REQUIRES}(\text{task}) \land l \notin \text{dom}(\text{Dictionary}) \} \)
7. for all \( m \in \text{Missing} \) do
8. \( \text{SOLVE}(m) \)
9. end for
10. \( \text{Inputs} \leftarrow \{ l | (l \to v) \in \text{Dictionary} \land l \in \text{REQUIRES}(\text{task}) \} \)
11. Dictionary \( \leftarrow \text{Dictionary} \cup \text{COMPUTES}(\text{task}) \cup \text{Inputs} \)
12. end if
13. end function

StenSAL: Explicit Stencil Algorithms in a Single Assignment Language
StenSAL allows for fast expression of explicit stencil algorithms in a functional style that is easily translated to REM. The following example is the expression of a task for solving a 3-point central stencil over a 1-dimensional domain:

```c
for all r in Result do
SOLVE(r)
end for

function SOLVE(label) then
if label \notin \text{dom}(\text{Dictionary}) then
  task ← PRODUCER(label)
  Missing ← \{ l | \text{task} \in \text{REQUIRES}(\text{task}) \land l \notin \text{dom}(\text{Dictionary}) \}
  for all m in Missing do
    SOLVE(m)
  end for
  \text{Inputs} ← \{ l | (l \to v) \in \text{Dictionary} \land l \in \text{REQUIRES}(\text{task}) \}
  Dictionary ← Dictionary ∪ COMPUTES(task) ∪ \text{Inputs}
end if
end function
```

StenSAL: Explicit Stencil Algorithms in a Single Assignment Language

The minimum dictionary size for a particular problem can be calculated as:

\[
|\Omega| = \alpha \prod_{k=1}^{n} (|T_{k+1}^\Gamma| + 1)
\]

\[
\Gamma = \max \{ \deg (\tau) | \tau \in \Omega \}
\]

\( T_{k+1}^\Gamma \) is the set of tasks when traversing the \( k \)th dimension of coordinate tuple \( c \) for sequencing index \( k+1 \), \( \Gamma \) is the retention factor, defined as the maximum in-degree of any task \( \tau \) in time step \( k+1 \), and \( \alpha \) is an additional term added for curve fitting.