AdaCore
The GNAT Pro Company

Panel on Ada & Parallelism: *Ada Container Iterators for Parallelism and Map/Reduce*

Tucker Taft
AdaCore Inc

Brad Moore
General Dynamics Canada

based on work with

Stephen Michell
Maurya Software

Luis Miguel Pinho
ISEP, Portugal

Ada-Europe 2016

Pisa, Italy

Outline

- **Motivation**
- **Parallel Blocks**
  - *Tasklet* model
- **Parallelized Loops over Arrays**
  - Might use *chunking*
- **Parallelized Chunked Loops over Containers**
- **Hyper-objects for Reduction**
- **Summary**

Why Parallel?
The *Right Turn* in Single-Processor Performance

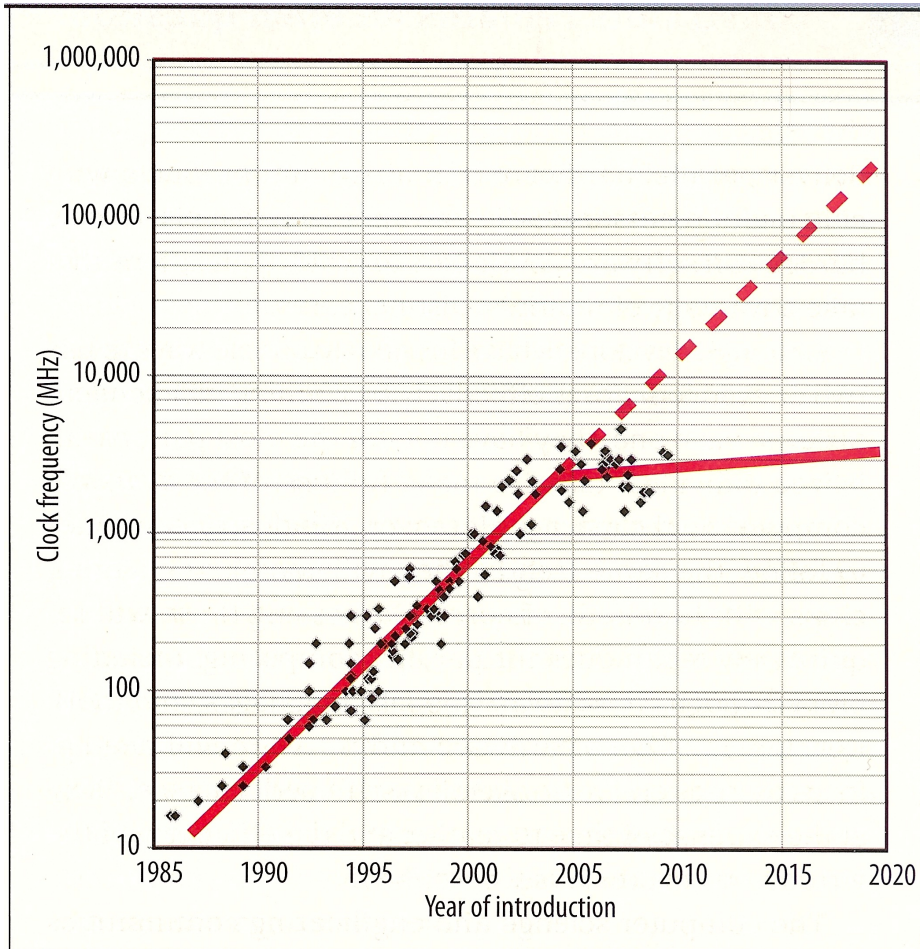


Figure 2. Historical growth in single-processor performance and a forecast of processor performance to 2020, based on the ITRS roadmap. A dashed line represents expectations if single-processor performance had continued its historical trend.

Courtesy IEEE Computer, January 2011, page 33.

Our Goal: Safe, Simple, Parallel Programming

- **What do we mean by "*parallel*" programming as opposed to "*concurrent*" programming?**
  - "*concurrent*" programming constructs allow programmer to *simplify* by using multiple threads to reflect the natural concurrency in the problem domain – heavier weight constructs OK
  - "*parallel*" programming constructs allow a programmer to *divide and conquer* a problem, using multiple (pico) threads (aka *tasklets)* to work in parallel on independent parts of the problem – constructs need to be light weight both syntactically and at run-time

Earlier Proposals for Parallel Ada Extensions

Parallel Blocks

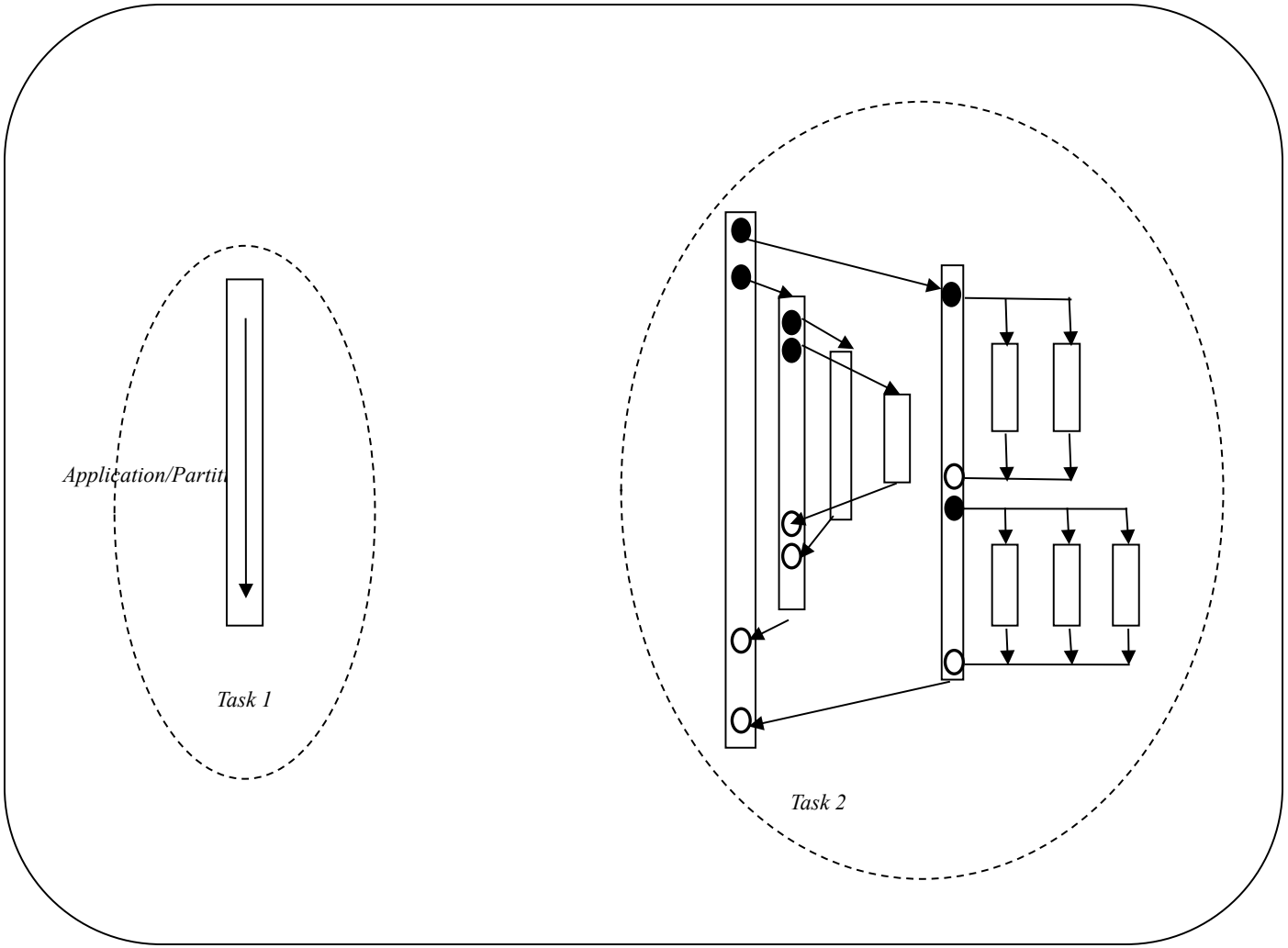
```
parallel_block_statement ::=
    parallel
      sequence_of_statements
    and
      sequence_of_statements
   {and
      sequence_of_statements}
    end parallel;
```

- Compiler *may* spawn each sequence as a separate *tasklet* but need not;
- May combine two, or run all sequentially

Example:

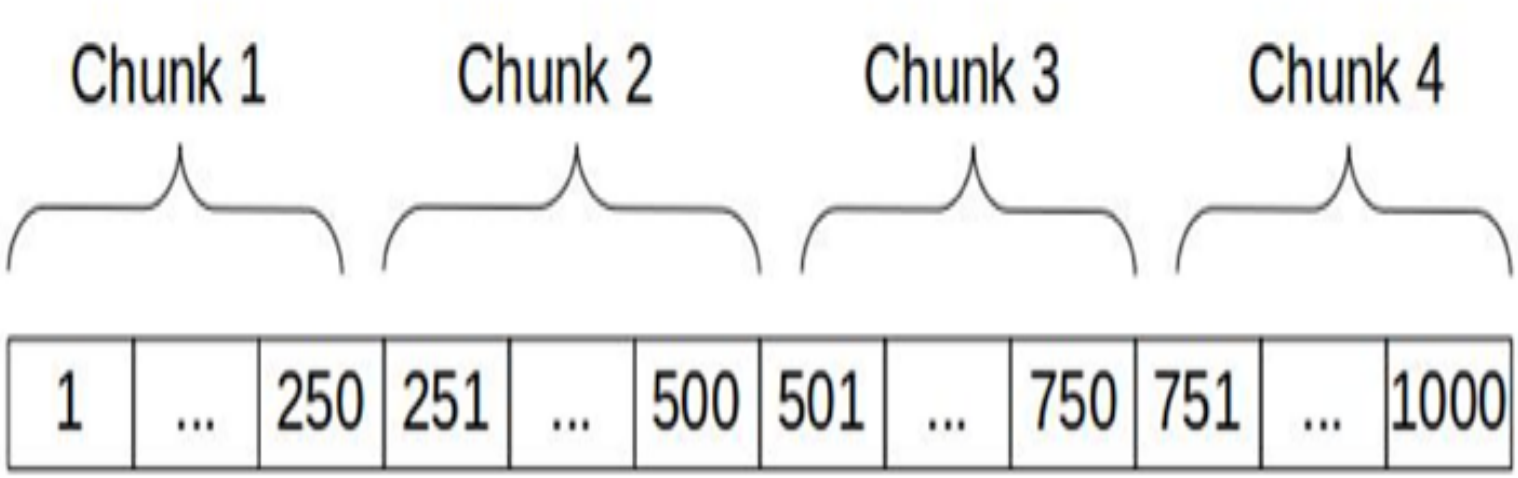
```
declare
   X, Y : Integer;
   Z : Float;
begin
   parallel
      X := Foo(100);
   and
      Z := Sqrt(3.14) / 2.0;
      Y := Bar(Z);
   end parallel; -- Implicit join point
   Put_Line("X + Y=" &
            Integer'Image(X + Y));
end;
```

Tasklet Model – Fork/Join parallelism *within* Ada task



Parallelized Loops – Might be split into "chunks"

```
for I in parallel 1 .. 1000 loop
   Process (I);
end loop;
```



Parallelized Loop with Parallel Arrays for Partial Reduction

```
declare
   Partial_Sum : array (parallel <>) of Float := (others => 0.0);
   Sum : Float := 0.0;
begin
   for I in parallel Arr'Range loop
     Partial_Sum(<>) := Partial_Sum(<>) + Arr(I);
   end loop;

   for J in Partial_Sum'Range loop
     Sum := Sum + Partial_Sum(J);
   end loop;

   Put_Line ("Sum over Arr = " & Float'Image (Sum));
end;
```

- Compiler chooses number of chunks because of “array (parallel <>)”
  - Partial_Sum automatically ends up with one element per chunk
- Partial_Sum(<>) selects appropriate element when inside loop

Automatic final reduction step using 'Reduced(...) attribute

```
declare
   Partial_Sum : array (parallel <>) of Float := (others => 0.0);
   Sum : Float := 0.0;
begin
   for I in parallel Arr'Range loop
     Partial_Sum(<>) := Partial_Sum(<>) + Arr(I);
   end loop;

   for J in Partial_Sum'Range loop
     Sum := Sum + Partial_Sum(J);
   end loop;

   Put_Line ("Sum over Arr = " & Float'Image (Sum));
end;
```

```
Put_Line ("Sum over Arr = " &
   Float'Image (Partial_Sum'Reduced));
```

New Proposals for Generalizing to Containers

Generalizing Chunked Parallel Iterators to Containers

```
for Elem of parallel (Num_Chunks)  My_Map loop
   Put_Line (Elem_Type'Image (Elem));
end loop;
```

```
declare
   Iter : Parallel_Iterator'Class := Iterate (My_Map);
   Cursors : Cursor_Array (1 .. Num_Chunks);
begin
   Split (Iter, Cursors);  -- Get starting points for each chunk
   for I in parallel Cursors'Range loop  -- One tasklet per chunk
      declare
         Curs : Cursor := Cursors (I);
         End_Curs : constant Cursor := (if I = Cursors'Last then No_Element else Cursors (I+1));
      begin
         while Curs /= End_Curs loop  --  Process the chunk sequentially
            declare
               Elem : Elem_Type renames My_Map (Curs);
            begin
               Put_Line (Elem_Type'Image (Elem));
               Curs := Iter.Next (Curs);
            end;
         end loop;
      end;
   end loop;
end;
```

Split Operation supported by Parallel_Iterator

- **Split operation defined for Iterator objects that implement the Parallel_Iterator interface:**

  (in addition to First and Next)

```
procedure Split (Object : Parallel_Iterator;
                 Cursors : out Cursor_Array);
```

- **Length of Cursors array determines number of chunks**
- **Split initializes Cursors array with starting points**
- **Chunks need not all be of the same size**
  - Split should divide overall iteration into *reasonably* similarly-sized sub-iteration chunks
  - For example, might break into chunks based on convenient sub-tree partitioning

Hyper-Objects for Reduction

- **We provide support for Map/Reduce over Containers using the notion of a *Hyper-Object***
- **Hyper-Object provides a vector for partial results with an element-per-chunk, plus a reduction operation**
- **Hyper-Object is indexable, using the chunk number as the index**
- **Hyper-Object has Reduce operation to produce a final value**

```
generic
   type Element_Type [(<>)] is private;
   Identity : in Element_Type;
   with function Reducer (Left, Right : Element_Type)
      return Element_Type;
package [Indefinite_]Hyper_Objects is …
```

Example of Hyper-Objects – Integer Sum and String Concatenate

```
declare
   package Int_Sums is
      new Hyper_Objects (String, Identity => 0, Reducer => "+");
   package Str_Cats is
      new Indefinite_Hyper_Objects (String, Identity => "", Reducer => "&");
   Hyp_Str : Str_Cats.Accumulator (Num_Chunks);
   Hyp_Int : Int_Sums.Accumulator (Num_Chunks);
begin
   for Elem of parallel (Num_Chunks) My_Str_Vec loop
        Hyp_Int(<>) := Hyp_Int(<>) + Elem'Length; -- Explicit reduction
        Hyp_Str.Update (<>, Elem); -- Reduction performed by Update
   end loop;
   declare  -- Do the final reductions across the chunks
      Combined_Str : String (1 .. Hyp_Int.Reduce) := Hyp_Str.Reduce;
   begin
      Put_Line (Combined_Str);
      . . .
   end;
end;
```

Summary

- **Support for Fine-Grained Parallelism can help make best use of new multicore hardware**
- **Parallel blocks and Parallel loops over arrays are the first step**
- **Supporting Parallel iteration over Containers is natural next step**
- **Proposed "Split" operation provides an array of cursors as starting points for chunk-based parallel iteration**
- **Proposed *syntactic sugar* uses Split operation**
- **Proposed "Hyper_Objects" generic supports user-defined chunk-based parallel reduction operation**
- **Presumes "<>" to refer to chunk index inside loop body**
- **More *syntactic sugar* to support reduction is TBD.**