An efficient implementation of persistent objects

Jacob Sparre Andersen

Jacob Sparre Andersen Research & Innovation

The 15th International Conference on Reliable Software Technologies – Ada-Europe 2010

Outline

Who Wouldn't Want Easy Orthogonal Persistence?

type T **is persistent** . . . ;

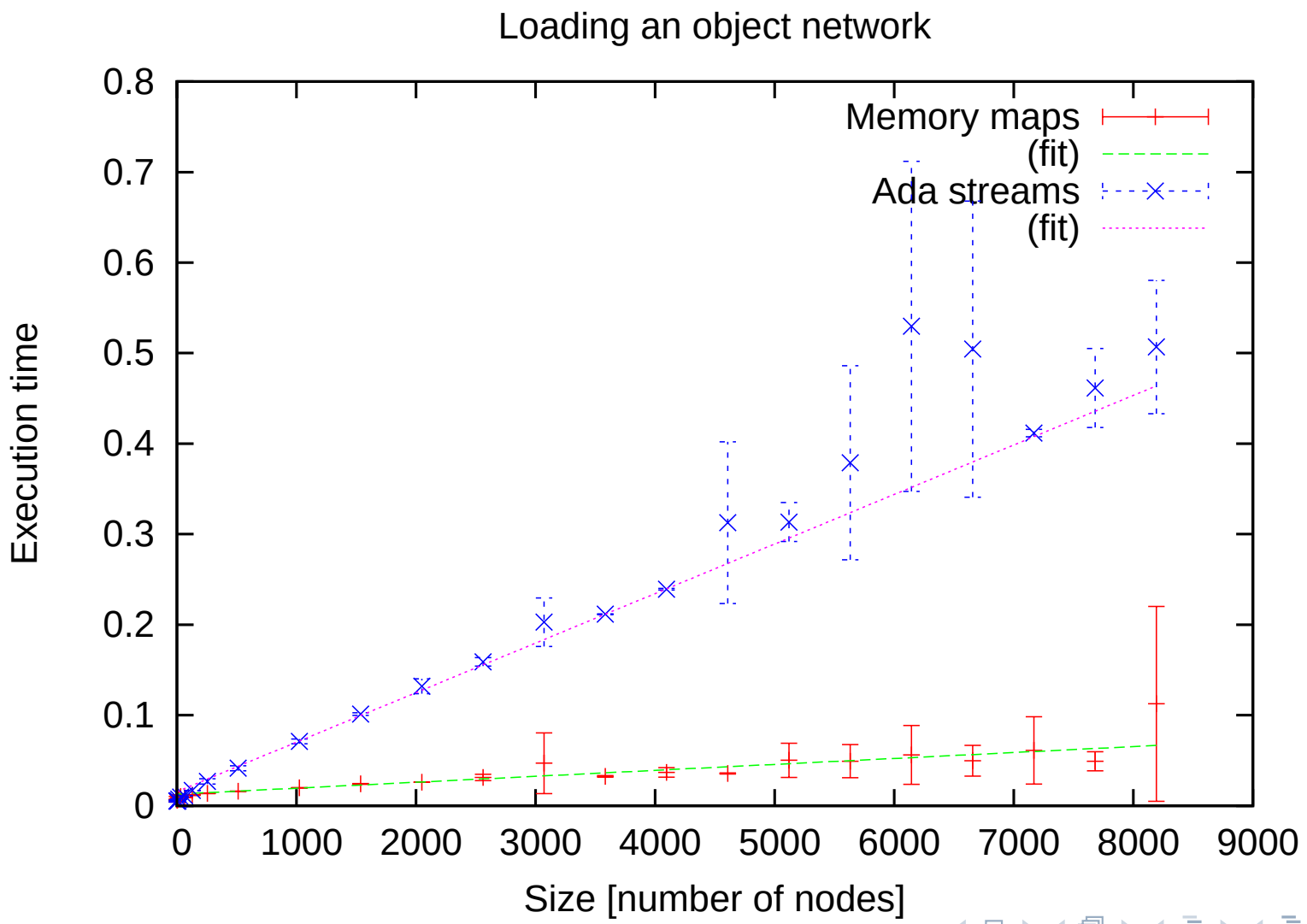
Unfortunately this is not legal Ada.

Almost Easy Orthogonal Persistence

```
type T is ...;
for T'Storage_Pool use ...;
```

This is legal Ada. – And with a small, fixed overhead it is enough to give us persistent objects.

Fast Orthogonal Persistence



The Idea

Memory Mapped Files and Storage Pools

There are two basic ideas behind this technique:

- Memory mapped files is an extremely fast I/O method.
- Ada storage pools allow us to control where in virtual memory (VM) dynamically allocated objects are stored.

Combined, this allows us to make dynamically allocated objects be located in a part of VM which is mapped to a file, and thus automaticaly stored.

package Persistent_Storage_Pool

```
package Persistent_Storage_Pool is
   type Instance is new System.Storage_Pools.Root_Storage_Pool with private;

   type Root_Object is abstract tagged null record;
   subtype Root_Class is Root_Object'Class;
   type Root_Name is access all Root_Class;

   Bad_Pool_Format : exception;
   Error           : exception;

   procedure Create
     (Pool         : in out Instance;
      As           : in     String;
      Initial_Value : in     Root_Class;
      Size         : in     System.Storage_Elements.Storage_Count);

   procedure Load (Pool : in out Instance;
                   From : in     String);

   function Root (Pool : Instance) return Root_Name;
private
```

Use example

```
with Persistent_Storage_Pool;
   ...
   Persistent : Persistent_Storage_Pool.Instance;
   ...
   type Some_Reference is access all Some_Class;
   for Some_Reference'Storage_Pool use Persistent;
   ...
   type Another_Reference is access all Another_Class;
   for Another_Reference'Storage_Pool use Persistent;
   ...
   type Root is new Persistent_Storage_Pool.Root_Object with ...;
   ...
   if ... then
      Create (Storage_Pool, ...);
   else
      Load (Storage_Pool, ...);
   end if;
   ...
   Do_Something (Storage_Pool.Root);
   ...
```

procedure Persistent_Storage_Pool.Create

```
procedure Create
   ...
      Pool.Address := Map_Memory (Length     => Pool.Size,
                                  Protection => Allow_Read +
                                                Allow_Write,
                                  Mapping    => Map_Shared,
                                  File       => Pool.File,
                                  Offset     => 0);
   ...
   declare
      Header : Persistent_Storage_Pool.Header;
      pragma Import (Ada, Header);
      for Header'Address use Pool.Address;
   begin
      Header := (Key       => Persistent_Storage_Pool.Key,
                 Address   => Pool.Address,
                 Allocated => Conversions.Storage (Header'Size),
                 Root      => null);
      ...
```

Ada 2017?

The introduction of memory layout randomization complicates the use of the presented technique.

Allowing overloading/substitution of the dereferencing operation would remove the need to reload the memory map at its original address.

- Can this be done by extending package System.Storage_Pools?
- Or would it require a larger change to the language?

Contact Information

- E-mail: `jacob@jacob-sparre.dk`
- Source code:
  `http://www.jacob-sparre.dk/persistence/`