Reflections on Dynamic Languages and Parallelism

David Padua
University of Illinois at Urbana-Champaign

Parallel Programming

- Parallel programming is
  - Sometimes for *Productivity*
    - Because some problems are more naturally solved in parallel. For example, some simulations, reactive programs.
  - Most often for *Performance*
    - Serial machines are not powerful enough
    - For scalability across machine generations. Scalability more important than absolute performance for microprocessor industry.
- Parallel programming can be
  - Implicit – Library/Runtime/compiler
  - Explicit – Threading, multiprocessing, parallel loops
    - Shared-memory
    - Distributed memory

Dynamic Languages

- Dynamic languages are for
  - Productivity. They "Make programmers super productive".
  - Not performance
- DLs are typically slow.
  - 10-100 (1000 ?) times slower than corresponding C or Fortran
- Sufficiently fast for many problems and excellent for prototyping in all cases
  - But must manually rewrite prototype if performance is needed.

Parallel Programming with Dynamic Languages

- Not always accepted by the DL community
  - Hearsay: javascript designers are unwilling to add parallel extensions.
  - Some in the python community prefer not to remove GIL – serial computing simplifies matters.
- Not (always) great for performance
  - Not much of an effort is made for a highly efficient, effective form of parallelism.
    - For example, Python's GIL and its implementation.
    - In MATLAB, programmer controlled communication from desktop to worker.

Parallel Programming with Dynamic Languages (cont.)

- Not (always) great to facilitate expressing parallelism (productivity)
  - In some cases (e.g. MATLAB) parallel programing constructs were not part of the language at the beginning.
  - Sharing of data not always possible.
    - Python it seems that arrays can be shared between processes, but not other classes of data.
    - In MATLAB, there is no shared memory.
  - Message passing is the preferred form of communication.
    - Process to process in the case of Python.
    - Client to worker in the case of MATLAB
    - MATLAB's parfor has complex design rules

Why Parallel Dynamic Language Programs ?

- There are reasons to improve the current situation
- Parallelism might be necessary for dynamic languages to have a future in the multicore era.
  - Lack of parallelism would mean no performance improvement across machine generations.
  - DLs are not totally performance oblivious. They are enabled by very powerful machines.
- When parallelism is explicit
  - For some problems it helps productivity
  - Enable prototyping of high-performing parallel codes.
  - Super productive parallel programming ?
- Can parallelism be used to close the performance gap with conventional languages ?

Detour. The real answer

- But, if you want performance, you don't need parallelism, all you need is a little

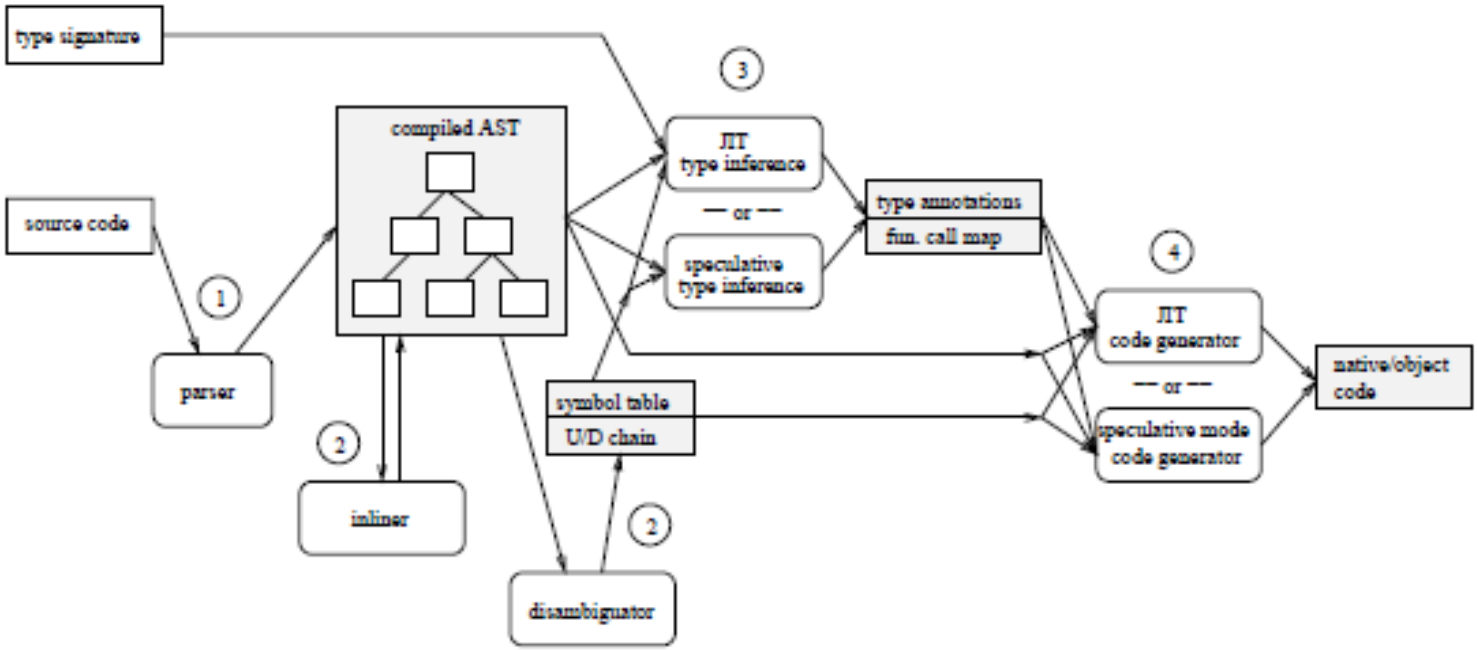
MaJIC: Compiling MATLAB for Speed and Responsiveness*

George Almási and David Padua
galmasi,padua@cs.uiuc.edu

Department of Computer Science
University of Illinois at Urbana-Champaign

PLDI'02, June 17-19, 2002, Berlin, Germany.

MaJIC


type signature
source code
1
parser
compiled AST
2
inliner
symbol table
U/D chain
2
disambiguator
3
JIT
type inference
— or —
speculative
type inference
type annotations
fun. call map
4
JIT
code generator
— or —
speculative mode
code generator
native/object
code

MaJIC Results

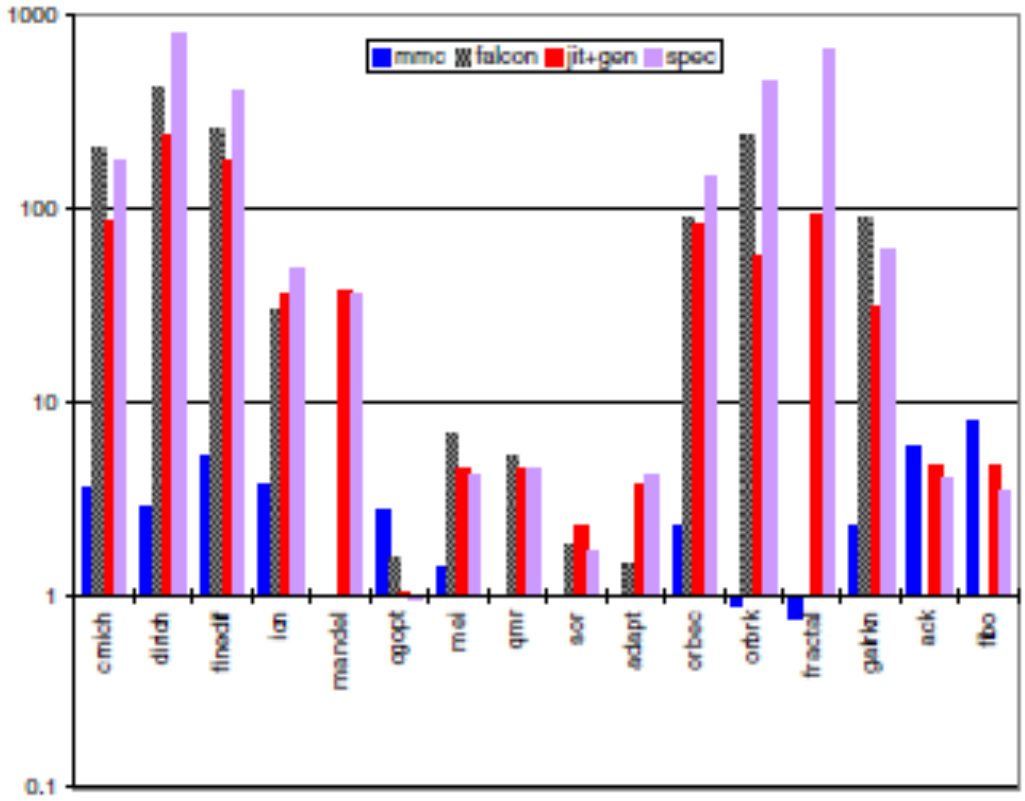


Figure 4: Performance on the SPARC platform

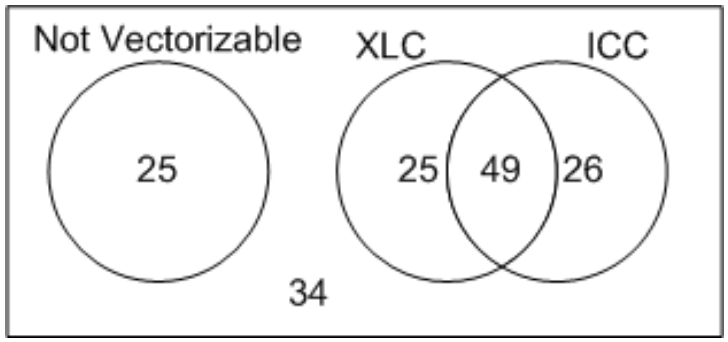
How to introduce parallelism
1. Autoparallelization

- Parallelism is automatic via compiler/interpreter
  - Perfect productivity
  - But the technology does not work in al cases. Not even for scientific programs.
  - Next slide shows an simple experiment on vectorization
    - Three compilers and a few simple loops.
    - Technology is not there not even for vectorization.

How to introduce parallelism
1. Autoparallelization (cont.)

Loops \ Compiler	XLC	ICC	GCC
Total	159		
Vectorized	74	75	32
Not vectorized	85	84	127
Average Speed Up	1.73	1.85	1.30

Loops \ Compiler	XLC but not ICC	ICC but not XLC
Vectorized	25	26



How to introduce parallelism
1. Autoparallelization (cont.)

- Would dynamic compilation improve the situation ?

- **NO**

How to introduce parallelism
2. Libraries of parallel kernels

- Again, programmer does not need to do anything
- Again, perfect from productivity point of view.
- This is the performance model of MATLAB.
  - Good performance if most of the computation were represented in terms of kernels.
  - Parallelism if most of the computation were represented in terms of parallel kernels.

- But not all programs can be written in terms of library routines.

How to introduce parallelism
3. (Data) Parallel Operators

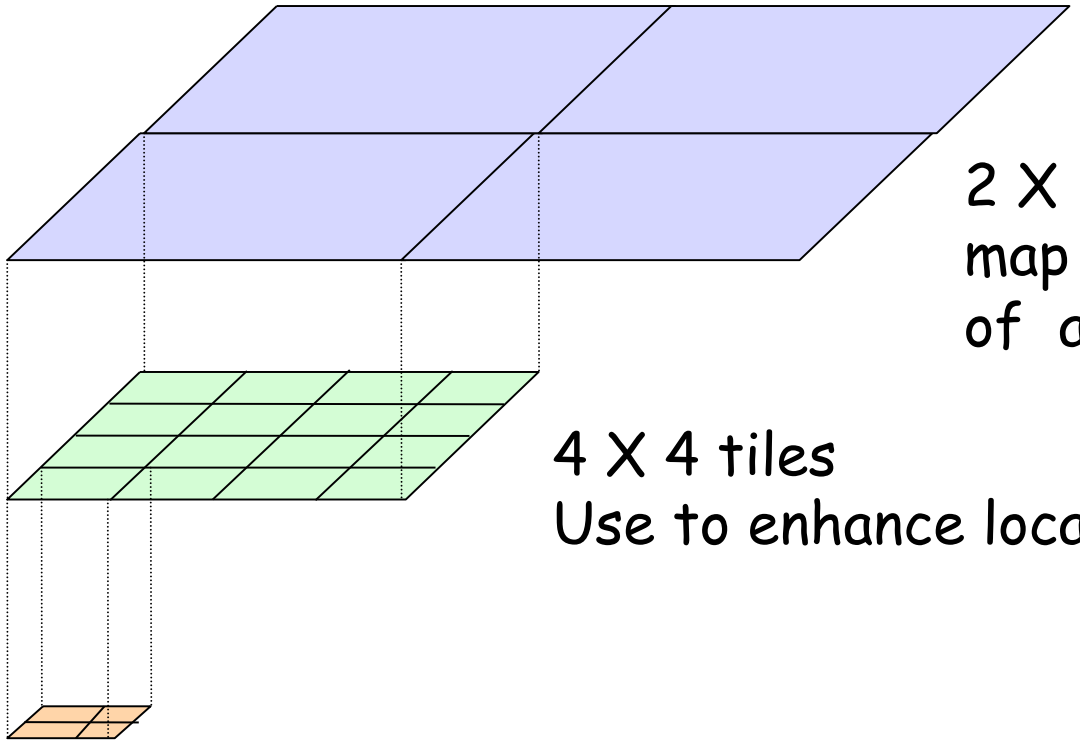
- Data parallel programs have good properties
  - Can be analyzed using sequential semantics
  - Parallelism is encapsulated
  - Can be used to enforce determinacy
  - For scientific codes, array notation produces highly compact and (sometimes) readable programs. Array notation introduced before parallelism (APL ca. 1960).
  - Recent (Re)Emergence of data parallel languages (e.g. Ct,)
- But they are explicitly parallel
  - More complex program development than their sequential counterpart.
- Not all forms of parallelism can be nicely represented
  - Pipelining
  - General task

3. (Data) Parallel Operators
Extending MATLAB: Hierarchically Tiled Arrays

- Blocking/tiling crucial for locality and parallel programming.
- Our approach makes tiles first class objects.
  - Referenced explicitly.
  - Manipulated using array operations such as reductions, gather, etc..

Joint work with IBM Research.
G. Bikshandi, J. Guo, D. Hoeflinger, G. Almasi, B. Fraguela, M. Garzarán, D. Padua, and C. von Praun. Programming for Parallelism and Locality with Hierarchically Tiled. *PPoPP*, March 2006.

3. (Data) Parallel Operators
Extending MATLAB: Hierarchically Tiled Arrays



2 X 2 tiles
map to distinct modules
of a cluster

4 X 4 tiles
Use to enhance locality on L1-cache

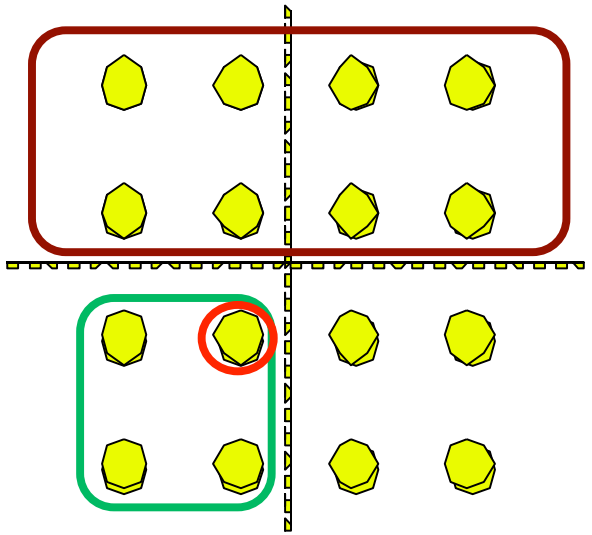
3. (Data) Parallel Operators
Extending MATLAB: Hierarchically Tiled Arrays

tiles

h{1,1:2}

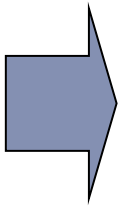
h{2,1}

h{2,1}(1,2)



3. (Data) Parallel Operators
Sequential MMM in MATLAB with HTAs

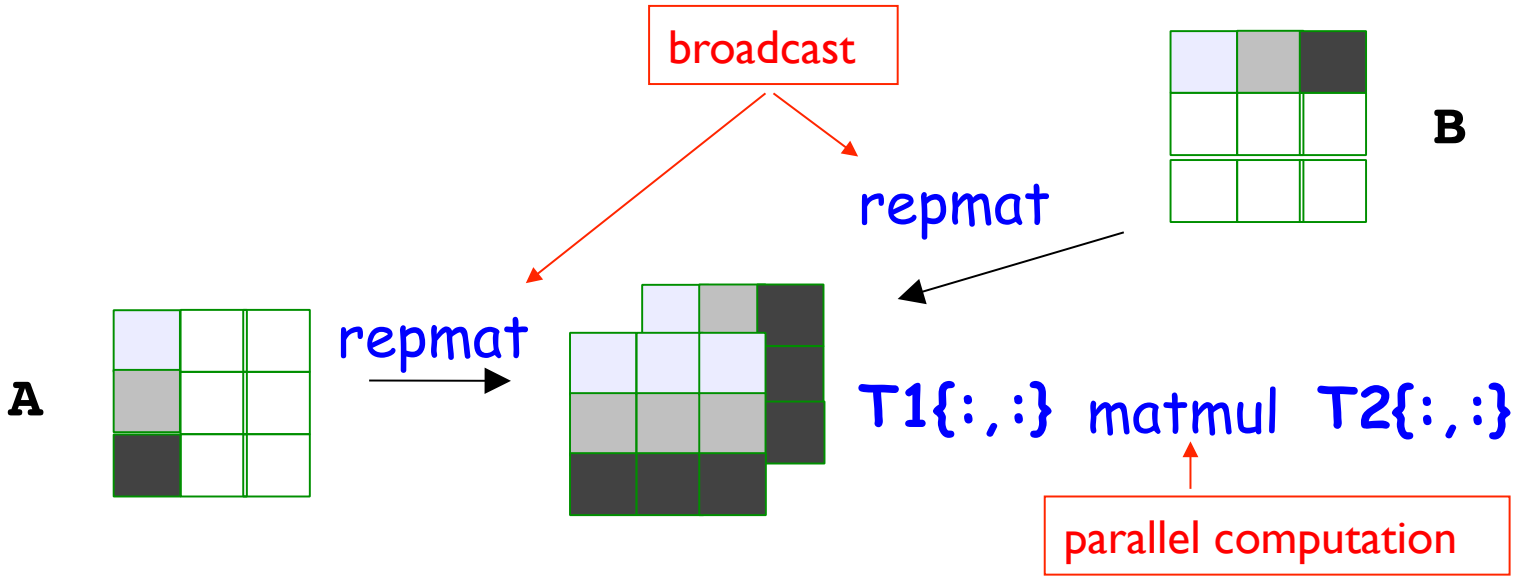
```
for I=1:q:n
   for J=1:q:n
      for K=1:q:n
         for i=I:I+q-1
            for j=J:J+q-1
               for k=K:K+q-1
                  C(i,j)=C(i,j)+A(i,k)*B(k,j);
               end
            end
         end
      end
   end
end
```



```
for i=1:m
   for j=1:m
      for k=1:m
         C{i,j}=C{i,j}+A{i,k}*B{k,j};
      end
   end
end
```

3. (Data) Parallel Operators
Parallel MMM in MATLAB with HTAs

```
function C = summa (A, B, C)
    for k=1:m
        T1 = repmat(A{:, k}, 1, m);
        T2 = repmat(B{k, :}, m, 1);
        C = C + matmul(T1{:,:} ,T2 {:,:});
end
```



How to introduce parallelism
4. General mechanisms

- This means
  - Fork, join
  - Parallel loops
  - Synchronization, semaphores, monitors
- Already in many languages. May need improvement, but no conceptual difficulty.
- Maximize flexibility/maximize complexity
- But, Good bye super productivity !
  - Race conditions
  - Tuning

Conclusions

- DLs of the future are likely to accommodate parallelism better than they do today.
- There are several possible approaches to introduce parallelism, but none is perfect.
- When (if?) parallelism becomes the norm:
  - Performance will have a more important role in DL programming
  - Therefore, at least for some classes of problems, super productivity will suffer.
- Advances in compilers, libraries, and language extensions will help recover some of the lost ground