Writing Better Charm Programs More Easily Using Charj

Aaron Becker
Illinois CS, Parallel Programming Lab

19 April 2011

What’s the matter with Charm?

Opaque semantics

What does this Charm code do?

```
w.compute();
x.compute();
y.compute();
z.compute();
```

Opaque semantics

What does this Charm code do?

```
w.compute(); // local method
x.compute(); // entry method
y.compute(); // sync entry method
z.compute(); // array broadcast
```

Opaque semantics

What does this Charm code do?

```
w.compute(); // blocking, local
x.compute(); // nonblocking, entry
y.compute(); // blocking, entry
z.compute(); // n invocations
```

Non-local semantic information

```
// A sync, expedited entry method
void foo::twist() { ... }

// A local method
void foo::shout() { ... }
```

Non-local semantic information

```
int n; // readonly variable
...
n = 17; // Ok if we’re in a
        // mainchare constructor.
        // Silent bug otherwise.
```

Stop repeating yourself

```
// foo.ci
entry void twist();

// foo.h
void twist();

// foo.cc
void foo::twist() { ... }
```

Limited Scope for Checking & Optimizations

- Most of your code is only seen by a C++ compiler
- No way to do lots of simple things:
  - observe messaging behavior
  - enforce Charm semantics
  - instrument or modify method implementations
- Moving more stuff into the translator sort of works (e.g. SDAG, accelerated entry methods), but it’s difficult, inflexible, and not very powerful.

Charj

Charm programs, but more productive

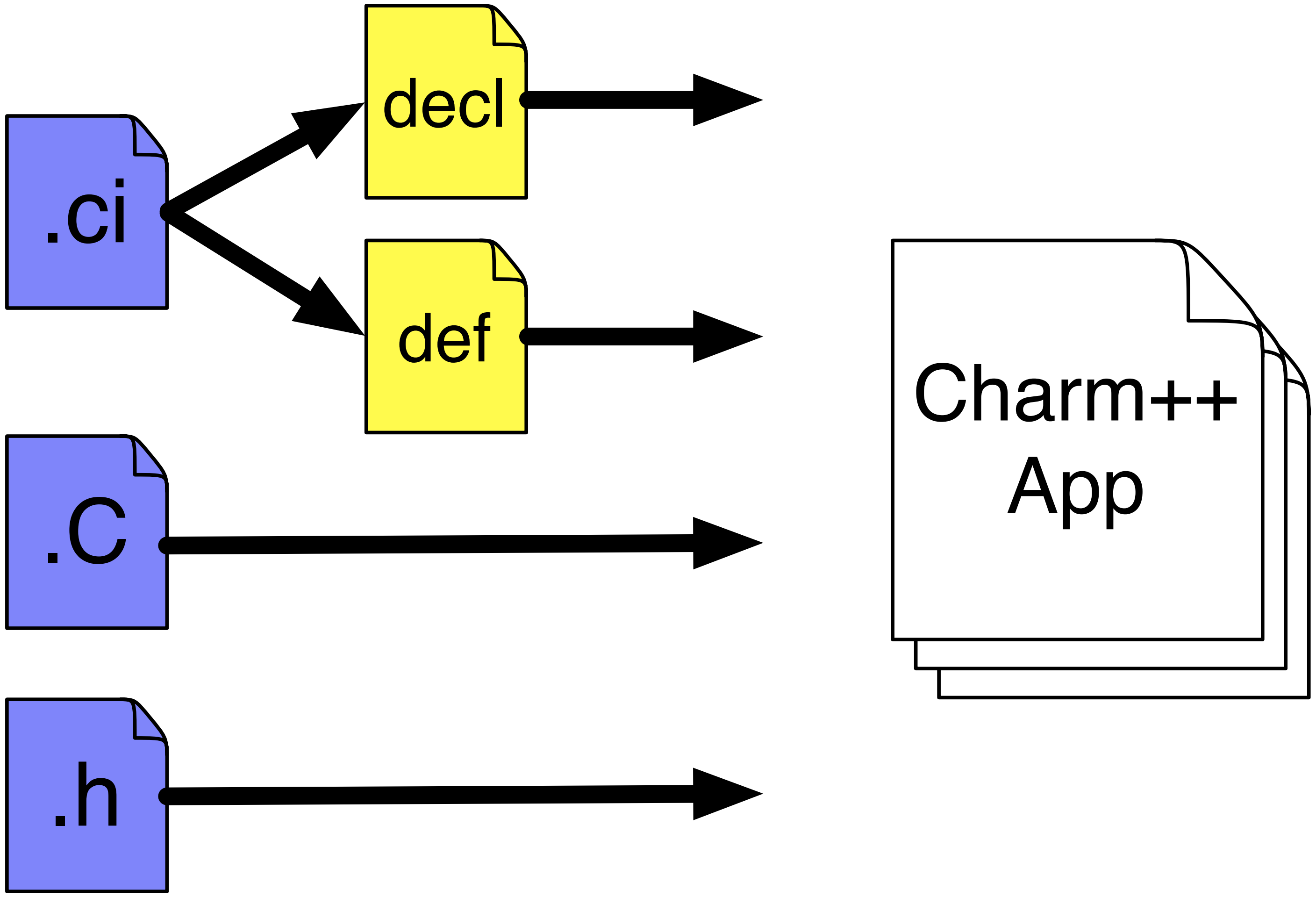
- Make syntax more meaningful
- Avoid boilerplate/automatic code
- Provide optimizations that are impossible at the library level
- Provide more safety
- Facilitate DSLs and “Little Languages”

Charj Infrastructure

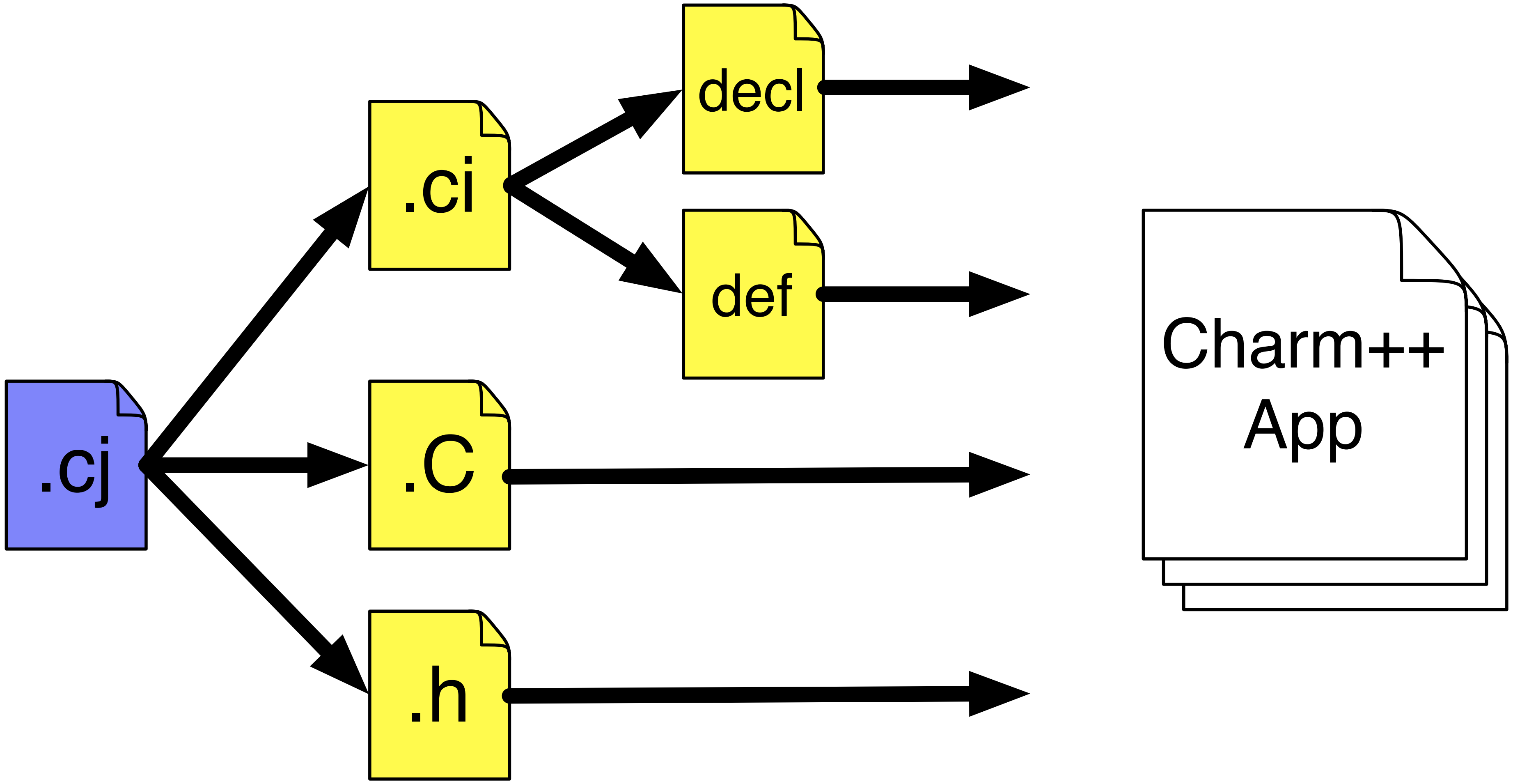
Technology

- ANTLR to lex, parse, and manipulate AST
- Simple Java/C#/D-inspired syntax
- Charj-specific libraries for things like arrays, ranges
- Translated to C++

Charm Translation/Compilation



Charj Translation/Compilation



Writing Clearer Programs

Charm-specific Syntax

- Use ‘@’ to indicate proxy operations/remote invocation

```
workerProxy.do_work(x, y, z,
       CkCallback(CkIndex_Manager::report_back));


vs


workerProxy@do_work(x, y, z, @Mananger.report_back);
```

Reductions in Charm

```
Worker::do_reduction() {
    int contribution = 1;
    contribute(1*sizeof(int), &contribution,
        CkReduction::sum_int);
}

Worker::reduction_done(CkReductionMsg* m) {
    int result = *((int*)m->getData());
    // ...
 }
```

Typed Reductions in Charj

```
Worker::do_reduction() {
    int contribution = 1;
    contribute(contribution, Reduction.sum);
}

Worker::reduction_done(int result) {
    // ...
}
```

Applying Simple Analysis

Packing and Unpacking

- Packing and unpacking data structures in Charm is mostly boilerplate

```
class Point3d {
    double x, y, z;
    void pup(PUP::er& p) { p|x; p|y; p|z; }
}
```

- Why should you have to write this code?
- What if you’re only going to use some of the class’s data on the receiving end?

Example: Computing Local Finite Element Solutions

```
class Element {
    // Geometry data
    vector coordinates, neighbors, normals;
    // Physics Data
    quadrature_info q;
    matrix basis, jacobian, boundary_data, cohesive_data;
}
smoothRegion(Element e, ... {...}
refineRegion(Element e, ...) {...}
coarsenRegion(Element e, ...) {...}
solveRegion(Element e, ...) {...}
```

How do we write entry methods?

```
smoothRegion(Element e, ...) // uses a subset of geom. data
refineRegion(Element e, ...) // a different subset of geom. data
solveRegion(Element e, ...) // uses a subset of phys. data
```

It would be wasteful to pack all that unneeded data

```
smoothRegion(SmoothMessage)
refineRegion(RefineMessage)
coarsenRegion(CoarsenMessage)
solveRegion(SolveMessage)
```

Lots of boring packing/unpacking code.

Creating Custom Pack/Unpack Code

```
entry void smoothRegion(Element e) {
    for each neighbor in e.neighbors
        if can_smooth(neighbor)
            ...
}
```

Creating Custom Pack/Unpack Code

```
entry void smoothRegion(Element e) {
    for each neighbor in e.neighbors
        if can_smooth(neighbor)
            ...
}
```

- identify all possible reads to *e* (being conservative)
- generate a custom method to pack/unpack only what is needed for this particular entry method
- fall back on full pack/unpack code where needed

Improving Charm’s “Little Languages”

MSA: Globally Addressable Arrays

```
// arr is an MSA array
for (int i=0; i<x_dim; ++i) {
    for (int j=0; j<y_dim; ++j) {
        update(arr[i][j]);
    }
}
```

MSA: Globally Addressable Arrays

```
// arr is an MSA array
for (int i=my_x; i < my_x+tile_x; ++i) {
    for (int j=my_y; j < my_y+tile_y; ++j) {
        update(arr[i][j]);
    }
}

// Inside arr[i][j], we need to check if the array element is
// local, and fetch it if needed.
```

This access may be local or non-local

MSA: Globally Addressable Arrays

```
// A higher-performance approach
arr.ensure_region_local(my_x, my_y, my_x+tile_x, my_y+tile_y);
for (int i=my_x; i < my_x+tile_x; ++i) {
    for (int j=my_y; j < my_y+tile_y; ++j) {
        update(arr.unsafe_read(i, j));
    }
}

// Faster, but also uglier.
```

MSA: Globally Addressable Arrays

```
for (int i=my_x; i < my_x+tile_x; ++i) {
    for (int j=my_y; j < my_y+tile_y; ++j) {
        update(arr[i][j]);
    }
}

// What we want: simple, natural expression
// without sacrificing efficiency.
```

Opportunities for Improvement

- Embed SDAG in any method
- Enforce readability and writability rules in accelerated entry methods
- Easy embedding of DSLs in Charm apps
  - DivCon, a DSL devoted to divide-and-conquer problems
- Better, smarter threading support

It is possible to write simpler, more expressive programs without giving up performance.