RnD Project Report: Expressive Power of Spawned Concurrent Languages

Under Prof. R.K. Shyamasundar

Siddhartha Dutta(120040005), Paramdeep Singh(120050085)

May 4, 2016

1 Introduction

In this project we try to give an implementation of guarded select in Habanero-C with complete non-determinism. The project involved figuring out a way to install Habanero which in itself is a tedious task due to lack of documentation with respect to its various implementations as a library or language extension. Finally, we settled for the HCLib library.

2 Implementation of guarded select

2.1 The API

We define the following struct whose object users create in order to create a single guarded select block. It is coded as follows:

Listing 1: The guarded_select structure

```
struct guarded_select{
private:
    int NUM_THREADS;
    int opt;
    int synchronizer1;
    int synchronizer2;
    struct ddf_st *ddf_list_cond[2];
    pthread_t thread[MAX_THREADS];

    pthread_mutex_t lock;

    int indices[MAX_THREADS];
    int indhelper[MAX_THREADS];

    void (*glguardif[MAX_THREADS])();
    void (*glguardthen[MAX_THREADS])();
};
```

All the member variables are private with none required to be accessed directly from user programs. Users just need to create objects of this struct in the manner below and pass its pointer to helper functions.

Listing 2: Initialization of a guarded_select structure object

```
struct guarded_select *g = malloc(sizeof(struct guarded_select));
```

Since C is not object oriented inherently, we emulate a constructor and execute function through the following functions that takes the guarded_select object's pointer as input.

1. This functions as a constructor in the tradition C++ sense.

   ```
   void guarded_select_construct (int num_threads,
           void (*guardif[])(), void (*guardthen[])(),
           struct guarded_select* g)
   ```

   It takes as arguments:

   (a) int num_threads: An integer representing the number of guarding conditions in our guarded select block.

   (b) void (*guardif[])(): An array of function pointers of size num_threads. Each function pointer points to a function that carries out the tasks expected to be carried out by the corresponding guarding condition.

   (c) void (*guardthen[])(): An array of function pointers of size num_threads. Each function pointer points to a function that carries out the tasks expected to be carried out by the block of code if its corresponding guarding condition is selected(i.e. the booleans in the guarding conditions were all true and it was possible for it to rendezvous with all processes that it was interested in communicating with)

   (d) struct guarded_select* g: A pointer to the guarded select object that we created above.

2. It is a blocking function that executes the whole guarding select block.

   ```
   void helper(int argc, char **argv,
           struct guarded_select* g)
   ```

   It takes as arguments:

   (a) int argc: The integer argument that main received indicating the number of command line arguments. The argc from main can be passed as-is.

   (b) char **argv: A pointer to the array of character strings that main received indicating the arguments that were typed onto the terminal. The argv from main can be passed as-is.

   (c) struct guarded_select* g: A pointer to the guarded select object that we created above.

2.2 Implementation

The guarded select construct was implemented in the following way:

1. We first create a random permutation of first n numbers using a random seed(here n denotes the number of guarding conditions).

   Listing 3: guarded_select_construct function

   ```
   void guarded_select_construct (int num_threads, void (*guardif[])(), void
        (*guardelse[])(), struct guarded_select* g) {

       g->synchronizer1 = 1;
       g->synchronizer2 = 1;
       pthread_mutex_init(&(g->lock), NULL);
       g->NUM_THREADS = num_threads;
       int i,j;
   ```

```
    for(i=0; i<g->NUM_THREADS; i++){
        g->glguardif[i]=guardif[i];
        g->glguardelse[i]=guardelse[i];
    }

    for(i=0; i<g->NUM_THREADS; i++){
        srand(time(NULL));
        int tmp = rand() % (g->NUM_THREADS-i);
        for(j=0; j<g->NUM_THREADS; j++){
            if(g->indhelper[j]==1 && j<=tmp)
                tmp++;
        }
        g->indices[i]=tmp;
        g->indhelper[tmp]=1;
    }
}
```

2. Then we spawn n threads corresponding to the n guarding conditions using async() function calls in the order of the permutation generated earlier.

Listing 4: helper function

```
void helper(int argc, char ** argv, struct guarded_select* g){

    hclib_init(&argc, argv);

    struct ddf_st ** ddf_list = (struct ddf_st **) malloc(sizeof(struct
        ddf_st *) * 2);
    ddf_list[0] = ddf_create();
    ddf_list[1] = NULL;

    void * arg [2];
    arg[0] = ddf_list;
    arg[1] = g;

    async(p0, arg, NULL, NULL, NO_PROP);
    start_finish();
    async(pf, NULL, ddf_list, NULL, NO_PROP);
    end_finish();
}
```

Listing 5: p0 function

```
void p0(void **arg1) {
    struct guarded_select* g = (struct guarded_select*) (arg1)[1];
    int successful_option = -1;

    int *argue = (int*) malloc(g->NUM_THREADS*sizeof(int));


    for(int i=0; i<g->NUM_THREADS; i++) {
        int j=g->indices[i];
        argue[j] = j;
        void **argtmp = malloc(2*sizeof(void*));
        argtmp[0]=&argue[j];
        argtmp[1]=g;
        pthread_create(&(g->thread[j]), NULL, &pi, argtmp);

    }

    g->synchronizer1 = 0;
    while(g->synchronizer2);
    successful_option = g->opt;
```

```
    for(int i=0; i<g->NUM_THREADS; i++) {
        if(successful_option==i) {
            (* g->glguardelse[successful_option])();
            printf("%d was called\n", i);

            int * value = (int *) malloc(sizeof(int)*1);
            *value = i;
            struct ddf_st ** argument = (struct ddf_st **) (arg1)[0];
            ddf_put(argument[0], value);
        }
    }
}
```

3. Now out of the n conditions, the one which becomes true first tries to attain a mutex lock. If multiple conditions become true together, one of them gets the lock randomly. Without using the permutation for spawning the threads, it was noted that the thread that was spawned earlier got the lock. Hence the required permutation was required to nullify this bias.
4. Obtaining this lock ensures that other threads won't execute beyond this point.
5. Now this thread which obtained the lock kills all the other n-1 threads.
6. Subsequently the "then function" corresponding to this thread is executed.

Listing 6: process i function

```
void pi(void **arg) {
    struct guarded_select* g = (struct guarded_select*) (arg)[1];

    int *arg1 = (int*) (arg)[0];
    int my_id = *arg1;
    printf("called %d\n", my_id);
    while(g->synchronizer1);
    (* g->glguardif[my_id])();

    pthread_testcancel();
    pthread_mutex_lock(&(g->lock));
    pthread_testcancel();
    for(int i=0; i<g->NUM_THREADS; i++) {
        if(i!=my_id) {
            printf("killing %d\n", i);
            pthread_cancel(g->thread[i]);
        }
    }
    printf("calling %d\n",my_id);
    pthread_mutex_unlock(&(g->lock));
    g->opt = my_id;
    g->synchronizer2 = 0;
}
```

2.3 Usage

Usage is as simple as declaring two arrays of function pointers of size equal to number of guarding conditions one wants in their guarded select block. The individual elements in the array must then be initialized to appropriate function pointers. Then a guarded_select object is created. Then guarded_select_construct function is called and finally the helper function is called.

Listing 7: guarded_select usage example

```
void if1(){while(1);}
```

```
void if2(){while(1);}
void if3(){;}

void else1(){printf("0 called\n");}
void else2(){printf("1 called\n");}
void else3(){printf("2 called\n");}

int main (int argc, char ** argv) {
    const int NUM_CONDITIONS = 3;
    void (*guardif[NUM_CONDITIONS])();
    guardif[0] = if1;
    guardif[1] = if2;
    guardif[2] = if3;

    void (*guardthen[NUM_CONDITIONS])();
    guardthen[0] = else1;
    guardthen[1] = else2;
    guardthen[2] = else3;

    struct guarded_select *g=malloc(sizeof(struct guarded_select));
    guarded_select_construct(NUM_CONDITIONS,guardif,guardthen, g);
    helper(argc, argv, g);
}
```

Its equivalent form in the traditional notation would be

Listing 8: Above code in traditional notation

```
if
  while(1) -> printf("0 called\n");
  while(1) -> printf("1 called\n");
  true     -> printf("2 called\n");
fi.
```

In the example above, we consider a guarded_select block with 3 conditions. The first function array pointer is declared as

```
void (*guardif[NUM_CONDITIONS])();
```

and then each of its elements are initialized to point to a function that executes the code representing the guarding condition.

The first two conditions involve an infinite while loop and will never get satisfied while the third one will trivially get satisfied.

The next function array pointer is declared as

```
void (*guardthen[NUM_CONDITIONS])();
```

which is again of size 3 and each of its element points to a function that executes code that was supposed to run had its guard been true and selected to execute.

We then create a struct guarded_select object and subsequently pass it to the guarded_select_construct and helper function.

The function guarded_select_construct is given the arguments NUM_CONDITIONS because there that many guarding conditions, the pointers to the two arrays guardif and guardthen created above and g, the pointer to our guarded_select struct.

The function helper is passed on argc and argv obtained from main and g, the pointer to our guarded_select struct.

3 Appendix

3.1 Emails

1. Hi Siddharta,

Nice to e-meet you. Hope all is well at your end.

The exit function defined in the paper is a member method of actors, please see

https://www.cs.rice.edu/ vs3/hjlib/doc/edu/rice/hj/api/HjActor.html The exit method sets an internal flag in the actor which causes the actor to stop it from processing subsequent messages, however it does not cause the current message processing to stop.

Also, our benchmarks code are available as part of the Savina benchmark suite:

https://github.com/shamsmahmood/savina If you wish access to the HJlib source code, my advisor Vivek Sarkar (cc-ed in this email) would be the person to talk to.

With regards to pre-emption we have another related work, The Eureka Programming Model for Speculative Task Parallelism, presented in ECOOP (http://2015.ecoop.org/track/research-track#event-overview). This model supports the preemption of tasks using the support of delimited continuations.

Let us know if you have more questions.

Best regards, Shams.

On Feb 8, 2016, at 9:33 AM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote:

Sir

We are working under Prof. R. K. Shyamasundar at IIT Bombay. As part of our project, we read your paper "Integrating Task Parallelism with Actors" which speaks about a exit() function call to terminate the actor. The code snippet presented in Fig. 4 also uses it. Since we are looking at the pre-emptive capabilities of languages such as Habanero we tried calling the exit function. However, we realized that the official versions of Habanero-C and java do not have this function defined.

We would be glad if you could provide us with the source code to your extended version of Habanero-java that you used for the purpose of the concerned paper. We would like to know more about it whether it support strong or weak pre-emption.

Looking forward to your quick reply.

Regards Siddhartha Dutta

2. Siddhartha,

   I'm still getting mixed messages, so let's clarify. What you talked to Shams about (with the actor support) was *HJlib*. What you downloaded and are showing me error messages for is *Habanero Java*, a defunct project which neither I nor anyone else in the group supports and which does not support actors or the exit() method you mentioned. It sounds like you are actually interested in HJlib, not Habanero Java. And therefore working on getting the thing you downloaded to build would be wasted effort, because it doesn't have what you are looking for. We need to be *very* precise and clear in the language about this, Habanero Java and HJlib are two completely separate and independent entities. There is no support for using Habanero Java.

   Does that all sound correct?

   Also, could you clarify what you mean by pre-emptive capabilities? That could mean many things. HClib certainly has the ability to suspend/preempt running tasks and uses that in its scheduling, so I want to understand the differentiation here.

   Max

   On Feb 11, 2016, at 7:35 AM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote:

   Hi Max

   Thanks for the quick reply.

Actually the subject was misleading as I just replied to your previous message. I have moved over phasers and have found an alternative.

What I was looking for was the exit() function mentioned in one of Shams papers. He mentioned the following to me in one of his replies.

"The exit function defined in the paper is a member method of actors, please see

https://www.cs.rice.edu/ vs3/hjlib/doc/edu/rice/hj/api/HjActor.html The exit method sets an internal flag in the actor which causes the actor to stop it from processing subsequent messages, however it does not cause the current message processing to stop."

I didn't find any pre-emptive capabilities in Habanero-C. Please correct me if I am wrong.

So, I downloaded Habanero-Java 1.3.1 from the link at

https://wiki.rice.edu/confluence/display/HABANERO/HJ-Download. Its INSTALL file refers to a HelloWorld file to be found at examples/HelloWorld. However, there's no HelloWorld directory within the examples directory. I would appreciate it if you could make the file available.

I tried some HelloWorld examples presented at some other download links. They lead to the following error on both mac and ubuntu.

****Error:

/Users/Siddhartha/Documents/code/rec/www.cs.rice.edu/ vs3/hjlib/code/

edu/rice/hj/example/comp322/labs/lab1/HelloWorldError.hj:26:28:26:28:0:-1:1: parsing terminated at this token

/Users/Siddhartha/Documents/code/rec/www.cs.rice.edu/ vs3/hjlib/code/

edu/rice/hj/example/comp322/labs/lab1/HelloWorldError.hj:1: Unable to parse HelloWorldError.hj. 1 error. compilation failed: Polyglot Front-End error

Thanks Siddhartha Dutta

On 2016-02-11 18:35, Max Grossman wrote: Hi Siddhartha,

Let me just clear up some nomenclature first:

Habanero Java - language-based extension to the Java programming language with parallel constructs Habanero C - language-based extension to the C programming language with parallel constructs HJlib - parallel programming library for the JVM HClib - parallel programming library for native C/C++ programs

If you're interested in the language options (Habanero Java/Habanero C), there isn't a maintainer I can point you to because those projects aren't being maintained. For the most part, those projects are defunct at this point. If you're interested in HJlib, Shams and I are the right people to be talking to. Shams is the creator and maintainer of HJlib, but he also recently graduated and took a full-time job which means he may have less and less time to work on it. If you're interested in HClib, I'm the best person to be talking to.

We're always open to implementing new constructs in HClib (e.g. things that are in HJlib but not in HClib), but we would first have to understand what the need is for that particular construct. For example, phasers have been supported in HClib in the past but we ripped them out because no one was using them and there was always a better option, at least for the applications we target. If there is a parallel pattern that is uniquely suited to phasers in some app that you're looking at, that would be a reason to think about moving them back in.

Max

On Feb 11, 2016, at 6:55 AM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote:

Hi Max

Could you get me in touch with someone who is maintaining the Habanero-Java project. The downloads page points to Vincent Cave but I think he has left the project and my mails to him are bouncing. Actually, in my conversation with Shams Imam I found out that some constructs that we require and that Shams implemented are only present in Habanero Java and are not present in Habanero C.

Thanks Siddhartha

On 2016-02-05 20:01, Max Grossman wrote: Siddhartha,

While the Habanero-Java and Habanero-C models share some features, they are fundamentally different models. We don't support isolated in HClib and have no plans to. In general, HClib does not support blocking locking or synchronization operations because once you start using them you lose most of your opportunity for achieving scalable parallelism. If you do have control or data dependencies in your computation, HClib would prefer that you express them explicitly (e.g. using promises) rather than implicitly using things like locks.

Of course, you're always free to combine other libraries with HClib, which would mean you could replicate isolated's functionality using pthreadmutext.

Thanks,

Max

On Feb 5, 2016, at 8:15 AM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote:

Hi Max

The paper "Habanero-Java: The new adventures of Old X10" describes an isolated construct but I could neither find its implementation in the source code of Habanero-C nor its usages in the test files. Am I missing something or is the construct yet to be implemented?

Thanks Siddhartha Dutta

On 2016-02-05 02:44, Max Grossman wrote: Siddhartha,

Gotcha, yes those instructions aren't exactly up to date. Phasers aren't still a part of the master branch at the moment. For the most part the install process is just a matter of running the install.sh script, if I was you I would focus more on just the simple install instructions at the top of README.md. Those are regularly maintained by myself, whereas that older page is not.

Yes, you are correct in that we used to have a custom compiler for the Habanero-C language that would allow for nicer syntax. However, the cost in man hours to maintain a whole compiler became a real problem for us and so we ended most of our support for it a couple of years ago (though I believe there are still a few research projects using it in the research group).

If you're looking for programmability similar to a language-based approach but without the need for a custom compiler, I'd recommend taking a look at our C++ APIs instead of our C APIs. Using C++ lambdas makes programming HClib much, much easier. Accessing the C++ APIs should simply be a matter of swapping out hclib.h for hclibcpp.h [1] and then using the functions declared in the hclib:: namespace instead of the functions that start with hclib*. In general, there is a one-to-one mapping of C APIs to C++ APIs, so it should be straightforward to port any code you already have for the C APIs to the C++ APIs.

Let me know if you have any more questions. If you run into any issues that you believe are bugs in HCLib, feel free to open an issue on the Github page so that we can take a look at it there.

Thanks,

Max

[1] https://github.com/habanero-rice/hcpp/blob/master/inc/hclibcpp.h

On Thu, Feb 4, 2016 at 2:07 PM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote: Hi Max

Thanks for your quick reply. I don't specifically require phasers. I was just following the instructions at http://habanero-rice.github.io/hclib/installation.html and stumbled upon it, so spent some time looking for the phaser-api.h file. It wasn't clear to me that it was no longer supported.

I would be glad if you could answer another query of mine. HCLib's page at http://habanero-rice.github.io/hclib/index.html claims that it is "A library implementation of the Habanero-C language". So, is there a non-library based implementation of the language too with may be a custom compiler in which the constructs can be written more easily?

For eg, https://wiki.rice.edu/confluence/display/HABANERO/Habanero-C mentions syntax of the form async [(place)] [IN (var1, var2, ...)] [OUT (var1, var2, ...)] [INOUT (var1, var2, ...)][AWAIT (ddf1, ddf2, ...)] [phased] Stmt

but with the library implementation I have to deal with

void ** argv1 = malloc(sizeof(void *) * 2); argv1[0] = malloc(sizeof(var1) *1); (argv1[0]) = done; argv1[1] = malloc(sizeof(var2) *1); (argv1[1]) = ddf1;

```
async(f1, argv1, NULL, NULL, NO_PROP);
```

Thanks Siddhartha Dutta

On 2016-02-05 00:42, Max Grossman wrote:

Siddhartha,

What is your interest in phasers in HClib? We have temporarily discontinued supporting them as we didn't have any large apps that were really making use of the support, so we didn't want to make on the maintenance burden without a good reason to do so. We can add that support back in, but it will take some effort on our end. Do you have a compelling application for phasers?

Thanks,

Max

On Thu, Feb 4, 2016 at 1:02 PM, Siddhartha Dutta ¡sdutta@cse.iitb.ac.in¿ wrote:

Hi

Installation of habanero c with phaser support(ORT backend) requires a phaserLib installation. But the phaser-api.h file is nowhere in the git repo of habanero or anywhere in the web. Could you please provide access to files required to compile the habanero c library with phaser support?

Regards Siddhartha Dutta

3. Hello Siddhartha, If I understand, the issue is that you do not want multiple writes to the DDF which is an error. Don't write to the DDF inside the EDT's created for the guarded condition. Instead of writing to the DDF inside the asyncs/EDT's created for guarded condition, you can create an event that depends on the completion of EDT's created for the guarded condition and once any of the guarded condition EDT finishes, this event is triggered (only once) which should enable another EDT that writes to the DDF(and also you can try to destroy other EDT's from here). Events require any of its pre-slot to be satisfied as opposed to EDT's which require all the pre-conditions to be satisfied.

Thank you Sriraj

On Mar 7, 2016, at 11:08 AM, Siddhartha Dutta ¡siddharthadutta@live.com¿ wrote:

Actually, I'm trying to implement a guarded select kind of functionality in hc-lib(a part of the Habanero project under Prof Vivek Sarkar at Rice univ). The habanero language provides constructs like async, finish, ddf etc.

async spawns the code written within it in a new edt and moves ahead with the previous program.

ddf is buffer of size 1 in which I can put an object once. One can make asyncs wait for a ddf put. Putting into a ddf twice is an error/ panic condition.

Guarded select is something like this: within a block there are several guard conditions each having a block of code associated with it which executes if the guard evaluates to true. If several of the guards are true, choice is made non-deterministically.

So, to implement guarded select, I start several asyncs each corresponding to each guarded condition and pass them the same ddf. After the code of the asyncs, I wait for anyone of the asyncs to put their async_id on the ddf. Now, as soon as one of the asyncs puts in their id, I want all the other asyncs to terminate.

Please tell me if some part of the above is unclear. I could share the code but its not elegant and I'm unaware of your knowledge of habanero.

Siddhartha

From: romain.e.cledat@intel.com To: ocr-user@eci.exascale-tech.com Date: Mon, 7 Mar 2016 18:41:03 +0000 Subject: Re: [OCR-User] Terminating/ killing edts in OCR

What is your use case? As I replied in another email, providing such a function would mean EDTs can, in a sense, synchronize while they are executing. It would also be hard to guarantee as the programmer does not really know when EDTs are running (just from what point they can potentially start).

I may be missing something though so definitely let me know what your use case is. It's possible we have another way of doing it already or need to add another mechanism to deal with it.

Thanks, Romain