## Referências Bibliográficas


[01] OUSTERHOUT, J.. **Scripting: higher level programming for the 21st century**. Computer, 31(3):23–30, Mar 1998. 1.1

[02] FOSTER, I.. **Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 1.1

[03] NIERSTRASZ, O.; BERGEL, A.; DENKER, M.; DUCASSE, S.; GAELLI, M. ; WUYTS, R.. **On the revival of dynamic languages**. In: Gschwind, T.; Aßmann, U., editors, PROCEEDINGS OF SOFTWARE COMPOSITION 2005, volumen 3628, p. 1–13. LNCS 3628, 2005. Invited paper. 1.1

[05] **The charm++ programming language manual**, 2000. `http://charm.cs.uiuc.edu/manuals/html/charm++/manual.html`. 3.1

[06] URURAHY, C.. **Um modelo de programação multilingual para aplicações geograficamente distribuídas**. PhD thesis, PUC-Rio, 2003. 1.3, 1.3.1

[07] KALE, L. V.; KRISHNAN, S.. **Charm++: Parallel Programming with Message-Driven Objects**. In: Wilson, G. V.; Lu, P., editors, PARALLEL PROGRAMMING USING C++, p. 175–213. MIT Press, 1996. 1.2

[08] IERUSALIMSCHY, R.; DE FIGUEIREDO, L. H. ; CELES, W.. **Lua 5.1 Reference Manual**. Lua.Org, 2006. 1.2

[09] WALKER, D.. **The design of a standard message passing interface for distributed memory concurrent computers**. Parallel Computing, 20:657–673, 1994. 2

[10] HILL, J. M. D.; MCCOLL, B.; STEFANESCU, D. C.; GOUDREAU, M. W.; LANG, K.; RAO, S. B.; SUEL, T.; TSANTILAS, T. ; BISSELING, R. H.. **BSPlib: The BSP programming library**. Parallel Computing, 24(14):1947–1980, 1998. 2

[11] ANANDA, A. L.; TAY, B. H. ; KOH, E. K.. **A survey of asynchronous remote procedure calls**. SIGOPS Oper. Syst. Rev., 26(2):92–109, 1992. 2

[12] HENNING, M.; VINOSKI, S.. **Advanced CORBA programming with C++**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 2

[13] **Python official website**. `http://www.python.org`. Acessado em Março 2008. 1.4

[14] KALE, L. V.; BHANDARKAR, M.; JAGATHESAN, N.; KRISHNAN, S. ; YELON, J.. **Converse: An Interoperable Framework for Parallel Programming**. In: PROCEEDINGS OF THE 10TH INTERNATIONAL PARALLEL PROCESSING SYMPOSIUM, p. 212–217, April 1996. 2.4

[15] SCHMIDT, D. C.. **Wrapper facade: a structural pattern for encapsulated functions within classes**. C++ Rep., 11(2):40–50, 1999. 4.2

[16] QUANTZ, J. J.; DUNKER, G.; BERGMANN, F. ; KELLER, I.. **The FLEX system**. Technical Report 124, Berlin, 1995. 4.4

[17] VOLKMAN, V. R.. **Bison: A GNU breed of YACC**. j-CUJ, 7(8), Aug. 1989. 4.4

[18] URURAHY, C.; RODRIGUEZ, N.. **Programming and coordinating grid environments and applications: Research articles**. Concurr. Comput. : Pract. Exper., 16(5):543–549, 2004. 1.3.1

[19] BRUNNER, R. K.; KALÉ, L. V.. **Handling application-induced load imbalance using parallel objects**. In: PARALLEL AND DISTRIBUTED COMPUTING FOR SYMBOLIC AND IRREGULAR APPLICATIONS, p. 167–181. World Scientific Publishing, 2000. 2.5

[20] IERUSALIMSCHY, R.. **Programming in Lua**. Lua.org, 2003. 4.5

[21] MILANÉS, A.. **Suporte de linguagens de programação para migração heterogênea forte**. PhD thesis, PUC-Rio, 2008. 4.5

[22] RICE, J. R.. **A metalgorithm for adaptive quadrature**. J. ACM, 22(1):61–82, 1975. 5.1

[23] CIRNE, W.; PARANHOS, D.; COSTA, L.; SANTOS-NETO, E.; BRASILEIRO, F.; SAUVE, J.; SILVA, F.; BARROS, C. ; SILVEIRA, C.. **Running**

**bag-of-tasks applications on computational grids: the mygrid approach**. Parallel Processing, 2003. Proceedings. 2003 International Conference on, p. 407–416, 6-9 Oct. 2003. 5.1

[24] AGARWAL, A.; LEVY, M.. **The kill rule for multicore**. In: DAC '07: PROCEEDINGS OF THE 44TH ANNUAL CONFERENCE ON DESIGN AUTOMATION, p. 750–753, New York, NY, USA, 2007. ACM. 1.1

[25] MASSINGILL, B.; MATTSON, T. ; SANDERS, B.. **Patterns for parallel application programs**, 1999. 5.1

[27] ASANOVIC, K.; BODIK, R.; CATANZARO, B. C.; GEBIS, J. J.; HUSBANDS, P.; KEUTZER, K.; PATTERSON, D. A.; PLISHKER, W. L.; SHALF, J.; WILLIAMS, S. W. ; YELICK, K. A.. **The landscape of parallel computing research: A view from berkeley**. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006. 1.1

[28] **Multicore means today's hpc is tomorrow's general purpose**. `http://www.regdeveloper.co.uk/2007/07/02/smith_parallel_coming/`. Acessado em Março 2008. 1.1

[29] BARRIENTOS, A.; RODRIGUEZ, N. ; SCHULZE, B.. **Managing jobs with an interpreted language for dynamic adaptation**. In: PROC. 3RD INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR GRID COMPUTING, Grenoble, Nov. 2005. 1.1

[30] MILLER, P.. **Parallel, distributed scripting with python**, 2002. 6.1.1

[31] VINOSKI, S.. **Rpc under fire**. Internet Computing, IEEE, 9(5):93–95, Sept.-Oct. 2005. 6.2

[32] MICROSYSTEMS, S.. **RPC: Remote procedure call protocol specification**, 1988. 6.2

[33] HUANG, C.; KALE, L. V.. **Charisma: Orchestrating migratable parallel objects**. In: PROCEEDINGS OF IEEE INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING (HPDC), July 2007. 6.1.2

[34] STROUSTRUP, B.. **The C++ Programming Language**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. 2

[35] JYOTHI, R.; LAWLOR, O. S. ; KALE, L. V.. **Debugging support for Charm++**. In: PADTAD WORKSHOP FOR IPDPS 2004, p. 294. IEEE Press, 2004. 3.1

[36] **Converse manual**, 2008. `http://charm.cs.uiuc.edu/manuals/html/converse/manual.html`. 3.1

# A
# Código de Testes

Neste apêndice está o código das aplicações utilizadas para os testes de desempenho discutidos no capítulo 5.

Como descrito anteriormente, foram feitas quatro implementações diferentes do algoritmo, sendo a primeira totalmente em C++, a segunda com o controlador em Lua e os trabalhadores em C++, a terceira inteiramente em Lua, e a quarta com ambos em Lua, mas os trabalhadores usando uma biblioteca em C para fazer os cálculos matemáticos necessários.

## A.1
## Primeira implementação

A seguir apresentamos o código da primeira implementação, inteiramente em C++.

### A.1.1
### Arquivo de Interface

```
mainmodule AQ
{
mainchare main {
entry main();
entry void setResult(double res);
entry void setPart(double left, double right);
entry void getWork(int id);
readonly CkChareID mainHandle;
};
group Calc {
entry Calc();
entry void putWork(double left, double right);
};
};
```

### A.1.2
### Arquivo de interface das classes do chare

```
#include <vector>
using namespace std;

class main : public Chare {
private:
typedef struct partition {
double left;
double right;
} Partition;

double totalRes;
vector<Partition> parts;
int sent, received;
CProxy_Calc calc;
double startTime;
public:
main(CkArgMsg *m);
static CkChareID mainHandle;

void setResult(double res);
void setPart(double left, double right);
void getWork(int id);
};

class Calc : public Group {
private:
CProxy_main mainchare;
public:
Calc();
void putWork(double left, double right);
};
```

### A.1.3
### Arquivo de implementação do chare

```
#include <stdio.h>
#include <stdlib.h>
```

```
// biblioteca padrão de Charm++
#include "charm++.h"

// interface do proxy - classe gerada
#include "AQ.decl.h"
// declaração da classe dos chares
#include "aq.h"
// código para cálculo da área
#include "calc.h"
// Implementação do proxy - classe gerada
#include "AQ.def.h"

// proxy global para o main chare
CkChareID main::mainHandle;

// inicia o chare principal. separa os intervalos para
// a bolsa de tarefas
main::main(CkArgMsg * m) {
mainHandle = thishandle;
totalRes = 0;
sent = received = 0;

double distance = (double) (MAXVAL - MINVAL) / PART_NUM;

for (int i = 0 ; i < PART_NUM ; i++) {
Partition part;
part.left  = distance * i + MINVAL;
part.right = distance * ( i + 1 ) + MINVAL;
parts.push_back(part);
}
CkPrintf("Iniciando os chares trabalhadores\n");
startTime = CkTimer();
calc = CProxy_Calc::ckNew();
};

// recebe o resultado do processamento de uma das partes
void main::setResult(double res) {
totalRes += res;
if (sent == ++received && parts.empty()) {
```

```
CkPrintf("Resultado do algoritmo: %lf . Resultado \
         preciso: %lf".\n",
totalRes, FUNCTION(MAXVAL) - FUNCTION(MINVAL));
CkPrintf("Tempo total: %10.6f\n", CkTimer() - startTime);
CkExit();
}
};

// insere um novo intervalo, resultado da sub-divisão de um
// intervalo grande
void main::setPart(double left, double right) {
Partition part;
part.left  = left;
part.right = right;
parts.push_back(part);
}

// funcao que recebe o pedido de um trabalhador por um
// intervalo
void main::getWork(int id) {
if (parts.empty()) {
calc[id].putWork(-1, -1);
return;
}
Partition part = parts.back();
parts.pop_back();
sent++;
calc[id].putWork(part.left, part.right);
};

// inicializa os trabalhadores
Calc::Calc() {
mainchare = (main::mainHandle);
if (CkMyPe() == 0)
for (int i = 0 ; i < CkNumPes() ; i++)
mainchare.getWork(i);
};

// callback chamada pela função main::getWork() para entregar
```

```
// um intervalo a um trabalhador
void Calc::putWork(double left, double right) {
if (left != -1) {
double minLocal  = left;
double maxLocal  = right;
double fMinLocal = FUNCTION(minLocal);
double fMaxLocal = FUNCTION(maxLocal);
double tArea     = (fMinLocal + fMaxLocal) *
   (maxLocal - minLocal) / 2;
double m  = (minLocal + maxLocal) /2;
double fM = FUNCTION(m);
double lArea = (fMinLocal + fM) * (m - minLocal) / 2;
double rArea = (fM + fMaxLocal) * (maxLocal - m) / 2;
double diff = tArea - (lArea + rArea);
if (diff < MAX_ERR) {
// envia o resultado para o main chare
mainchare.setResult(lArea + rArea);
} else {
// divide o intervalo em duas partes
mainchare.setPart(m, maxLocal);
// calcula a área do intervalo
double localRes = calcArea(minLocal, m, fMinLocal, fM, lArea);
mainchare.setResult(localRes);
}
mainchare.getWork(CkMyPe());
}
}
```



### A.1.4
### Implementação da lógica de cálculo da área

```
#ifndef TRAB4_H_
#define TRAB4_H_

#define FUNCTION(x) exp(x)

#define MAXVAL 15
#define MINVAL 0

#define PART_NUM 16
```

```
#define MAX_ERR 1e-16

double calcArea(double left, double right, double fLeft ,
                double fRight, double tArea) {
double result;
double m, fM, diff, lArea, rArea;

m  = (left + right)/2;
fM = FUNCTION(m);

lArea = (fLeft + fM) * (m - left) / 2;
rArea = (fM + fRight) * (right - m) / 2;

diff = tArea - (lArea + rArea);

if ( diff < MAX_ERR) {
result = lArea + rArea;
} else {
result =  calcArea(left, m, fLeft, fM, lArea);
result += calcArea(m, right, fM, fRight, rArea);
}
return result;
}
#endif /*TRAB4_H_*/
```



## A.2 Segunda implementação

Segue abaixo o código da segunda implementação, na qual o controlador é feito em Lua, e os trabalhadores em C++.

### A.2.1 Arquivo de interface

Como se pode observar no código abaixo, o arquivo de interface é igual ao da primeira implementação, com a diferença que o segundo tem o chare declarado como *Lua*.

```
mainmodule Part3 {
mainchare[lua "main.lua"] main {
entry main();
```

```
entry void setResult(double res);
entry void setPart(double left, double right);
entry void getWork(int id);
readonly CkChareID mainHandle;
};

group Calc {
entry Calc();
entry void putWork(double left, double right);
};
};
```

### A.2.2
### Interface dos trabalhadores

```
class Calc : public Group {
private:
CProxy_main mainchare;
public:
Calc();
void putWork(double left, double right);
};
```

### A.2.3
### Implementação dos trabalhadores

```
// Inicializa os trabalhadores
Calc::Calc() {
mainchare = (main::mainHandle);
if (CkMyPe() == 0)
                for (int i = 0 ; i < CkNumPes(); i++)
                        mainchare.getWork(i);
};

// Função chamada pelo chare controlador para
// entregar um intervalo para o trabalhador calcular
// a área
void Calc::putWork(double left, double right) {
if (left != -1) {
double minLocal  = left;
double maxLocal  = right;
```

```
double fMinLocal = FUNCTION(minLocal);
double fMaxLocal = FUNCTION(maxLocal);
double tArea     = (fMinLocal + fMaxLocal) *
                   (maxLocal - minLocal) / 2;

double m  = (minLocal + maxLocal) /2;
double fM = FUNCTION(m);
double lArea = (fMinLocal + fM) * (m - minLocal) / 2;
double rArea = (fM + fMaxLocal) * (maxLocal - m) / 2;

double diff = tArea - (lArea + rArea);

if (diff < MAX_ERR) {
mainchare.setResult(lArea + rArea);
} else {
double rRight, rLeft;
rLeft  = m;
rRight = maxLocal;

mainchare.setPart(rLeft, rRight);

double localRes = calcArea(minLocal, m, fMinLocal, fM, lArea);

mainchare.setResult(localRes);
}
mainchare.getWork(CkMyPe());
}
}
```



### A.2.4 Implementação do controlador em Lua

Abaixo temos o código do controlador em Lua.

```
local MAXVAL = 15
local MINVAL = 0
local PART_NUM = 16

main = {
ckNew = function()
```

```
totalRes = 0;
sent = 0;
received = 0;
parts = {}

distance = (MAXVAL - MINVAL) / PART_NUM;

for i = 1, 16 do
local left  = distance * (i - 1) + 0;
local right = distance * ( i ) + 0;
table.insert(parts, {left=left, right=right});
end

countTime = os.clock();
calc = charm.Calc();
end,

setResult = function(result)
totalRes = totalRes + result;
received = received + 1;
if sent == received and #parts == 0 then
print("Total elapsed time: ".. (os.clock() - countTime));
print("Resultado do algoritmo: "..totalRes);
end
end,

setPart = function(left, right)
table.insert(parts, {left=left, right=right});
end,

getWork = function(id)
if #parts == 0 then
calc[id].putWork(-1, -1);
return
end
local part = table.remove(parts)

sent = sent + 1;
```

```
local work = calc[id].putWork;
calc[id].putWork(part.left, part.right);
end
}
```

## A.3
## Terceira implementação

Abaixo apresentamos o código da terceira implementação, que é toda feita em Lua.

### A.3.1
### Implementação do Controlador

Como podemos ver, a implementação do controlador é a mesma que anteriormente, sem a necessidade de haver mudança pelo fato dos trabalhadores estarem sendo implementador em Lua.



```
local FUNCTION = math.exp
local MAX_ERR = 1e-16

function calcArea(left, right, fLeft , fRight, tArea)

  local result;
  local m, fM, diff, lArea, rArea;

  m  = (left + right)/2;
  fM = FUNCTION(m);

  lArea = (fLeft + fM) * (m - left) / 2;
  rArea = (fM + fRight) * (right - m) / 2;

  diff = tArea - (lArea + rArea);

  if  diff < MAX_ERR then
    result = lArea + rArea;
  else
    result =  calcArea(left, m, fLeft, fM, lArea);
    result = result + calcArea(m, right, fM, fRight, rArea);
  end
```

```
  return result;

end

Calc = {
ckNew = function()
if ck.mype() == 0 then
for i=1,ck.numpes() do
main.mainHandle.getWork(i-1);
end
end

end,

putWork = function(left, right)
if left ~= -1 then

local minLocal = left;
local maxLocal = right;

local fMinLocal = FUNCTION(minLocal);
local fMaxLocal = FUNCTION(maxLocal);
local tArea     = (fMinLocal + fMaxLocal) *
                  (maxLocal - minLocal) / 2;

local m  = (minLocal + maxLocal) /2;
local fM = FUNCTION(m);
local lArea = (fMinLocal + fM) * (m - minLocal) / 2;
local rArea = (fM + fMaxLocal) * (maxLocal - m) / 2;

local diff = tArea - (lArea + rArea);

if diff < MAX_ERR then
main.mainHandle.setResult(lArea + rArea);
else
local rRight, rLeft;
rLeft  = m;
rRight = maxLocal;
main.mainHandle.setPart(rLeft, rRight);
```

```
local localRes = calcArea(minLocal, m, fMinLocal, fM, lArea);

main.mainHandle.setResult(localRes);
end
main.mainHandle.getWork(ck.mype());
end

end
}
```

### A.3.2
### Implementação dos trabalhadores

Nesta implementação, temos todos os trabalhadores implementados em Lua, conforme o código abaixo, tendo todos os cálculos matemáticos feitos em Lua.



```
local MAXVAL = 15
local MINVAL = 0
local PART_NUM = 16

main = {
ckNew = function()
totalRes = 0;
sent = 0;
received = 0;
parts = {}

distance = (MAXVAL - MINVAL) / PART_NUM;

for i = 1, 16 do
local left  = distance * (i - 1) + 0;
local right = distance * ( i ) + 0;

table.insert(parts, {left=left, right=right});
end

countTime = os.clock();

calc = charm.Calc();
```

```
end,

setResult = function(result)

totalRes = totalRes + result;
received = received + 1;
if sent == received and #parts == 0 then
print("Tempo total: ".. (os.clock() - countTime));
print("Resultado do algoritmo: "..totalRes);
end
end,

setPart = function(left, right)
table.insert(parts, {left=left, right=right});
end,

getWork = function(id)
if #parts == 0 then
calc[id].putWork(-1, -1);
return
end
local part = table.remove(parts)

sent = sent + 1;
calc[id].putWork(part.left, part.right);
end
}
```



### A.3.3 Arquivo de interface de Charm++

Como se pode observar no código abaixo, o arquivo de interface é igual à da segunda implementação, com a diferença que ambos os chares são declarados como *Lua*.

```
mainmodule Part3
{
mainchare[lua "main.lua"] main
{
entry main();
```

```
entry void setResult(double res);
entry void setPart(double left, double right);
entry void getWork(int id);
readonly CkChareID mainHandle;
};

group [lua "calc.lua"] Calc
{
entry Calc();
entry void putWork(double left, double right);
};
};
```

## A.4 Quarta implementação

Na última implementação, temos ambos os chares implementados em Lua, mas com os trabalhadores utilizando uma lib implementada em C, que faz os cálculos matemáticos.

A interface e o controlador são iguais ao exemplo acima.

### A.4.1 Implementação dos trabalhadores

```
package.loadlib("./quad.so", "luaopen_quad")()

local FUNCTION = math.exp
local MAX_ERR = 1e-16
local int = 0

Calc = {
ckNew = function()
if ck.mype() == 0 then
for i=1,ck.numpes() do
main.mainHandle.getWork(i-1);
end
end
end,
putWork = function(left, right)
if left ~= -1 then
local minLocal = left;
```

```
local maxLocal = right;
local fMinLocal = FUNCTION(minLocal);
local fMaxLocal = FUNCTION(maxLocal);
local tArea     = (fMinLocal + fMaxLocal) *
                  (maxLocal - minLocal) / 2;

local m  = (minLocal + maxLocal) /2;
local fM = FUNCTION(m);
local lArea = (fMinLocal + fM) * (m - minLocal) / 2;
local rArea = (fM + fMaxLocal) * (maxLocal - m) / 2;
local diff = tArea - (lArea + rArea);

if diff < MAX_ERR then
main.mainHandle.setResult(lArea + rArea);
else
local rRight, rLeft;
rLeft  = m;
rRight = maxLocal;
main.mainHandle.setPart(rLeft, rRight);
local localRes = calc.calc(minLocal, m, fMinLocal, fM, lArea);
main.mainHandle.setResult(localRes);
end
main.mainHandle.getWork(ck.mype());
int = int + 1
else
ck.printstr("total: "..tostring(int))
end
end
}
```

O código da biblioteca em C fica como abaixo:

```
#include "lua.h"
#include "lualib.h"
#include "lauxlib.h"
#include <stdio.h>
#include <math.h>

#define FUNCTION(x) exp(x)
```

```
#define MAXVAL 15
#define MINVAL 0

#define PART_NUM 16

#define MAX_ERR 1e-16

static double calcArea(double left, double right, double fLeft,
                       double fRight, double tArea);
static int calcMinMax(lua_State* L){
double min = lua_tonumber(L, 1);
double max = lua_tonumber(L, 2);
double fMinLocal = FUNCTION(min);
double fMaxLocal = FUNCTION(max);
double tArea     = (fMinLocal + fMaxLocal) * (max - min) / 2;
lua_pushnumber(L,
    calcArea(min, max, fMinLocal, fMaxLocal, tArea));
return 1;
}
static int calc(lua_State* L){
double left   = lua_tonumber(L, 1);
double right  = lua_tonumber(L, 2);
double fLeft  = lua_tonumber(L, 3);
double fRight = lua_tonumber(L, 4);
double tArea  = lua_tonumber(L, 5);
lua_pushnumber(L,
    calcArea(left, right, fLeft, fRight, tArea));
return 1;
}
int luaopen_quad(lua_State* L){
luaL_reg libmethods[]={
  {"calc", calc},
  {"calcMinMax", calcMinMax},
  {NULL,   NULL}
};
luaL_register(L, "calc", libmethods);
return 1;
}
double calcArea(double left, double right, double fLeft,
```

```
                double fRight, double tArea){
double result;
double m, fM, diff, lArea, rArea;
m  = (left + right)/2;
fM = FUNCTION(m);
lArea = (fLeft + fM) * (m - left) / 2;
rArea = (fM + fRight) * (right - m) / 2;
diff = tArea - (lArea + rArea);
if ( diff < MAX_ERR){
result = lArea + rArea;
}
else{
result =  calcArea(left, m, fLeft, fM, lArea);
result += calcArea(m, right, fM, fRight, rArea);
}
return result;
}
```

# B
# Código do Façade

Neste apêndice apresentamos o código gerado de um façade para dois chares simples. O primeiro chare é um chare normal, e o segundo chare é um grupo. Apesar do código ainda poder sofrer otimizações para aumentar o desempenho da aplicação, percebemos que isso não teve um impacto muito grande no tempo de execução.

O arquivo de interface da aplicação neste exemplo é:

```
//////////////////////////////////////
// Arquivo: pgm.ci
mainmodule Prog {
  mainchare[lua "main.lua"] main {
    entry[lua] main();
    entry[lua] void setResult(int res);
    readonly CkChareID mainHandle;
  };
  group [lua "calc.lua"] calc {
    entry calc();
    entry [lua] void putWork(double left, double right);
  };
};
```

Através da execução do parser, temos o código gerado da implementação em C++ dos chares. A seguir apresentamos o código de declaração das classes dos chares:

```
//////////////////////////////////////
// Declaração
class main : public CBase_main{
  public:
    void setResult(double res);
    static CkChareID mainHandle;
    main (CkArgMsg *m);
```

```
  private:
    lua_State *L;
};
class Calc : public CBase_Calc{
  public:
    void putWork(double left, double right);
    Calc ();
  private:
    lua_State *L;
};
```

Como vemos, o código das classes gerado é bastante similar ao código dos chares apresentado ao longo desta dissertação. com a inclusão de uma variável `lua_State*`, que é um ponteiro para a máquina virtual de Lua.

A seguir apresentamos o código gerado pelo parser para a implementação das classes e funções auxiliares geradas pelo binding.



```
// Protótipo da função responsável pela instanciação do chare
// Calc de Lua.
static int _Lua_static_Calc(lua_State* L);
// Exporta a biblioteca charm, com as funções responsáveis por
// instanciar novos chares
static void registerChares(lua_State* L){
  lua_newtable(L);
  lua_pushstring(L, "Calc");
  lua_pushcfunction(L, &_Lua_static_Calc);
  lua_settable(L, -3);
  lua_setglobal(L, "charm");
}

// Declaração das funções exportadas para a biblioteca ck
int CkLua_printstr(lua_State* L) {
  if (!lua_isstring(L, 1)) return 0;
  const char *stringToPrint = lua_tostring(L, 1);
  CkPrintf("%s\n",stringToPrint);
  return 0;
}
int CkLua_mype(lua_State* L) {
```

```
  if (lua_gettop(L) > 0) return 0;
  lua_pushinteger(L, CkMyPe());
  return 1;
}
int CkLua_numpes(lua_State* L) {
  if (lua_gettop(L) > 0) return 0;
  lua_pushinteger(L, CkNumPes());
  return 1;
}
luaL_reg CkLua_MethodsDefault[]={
  {"printstr",    CkLua_printstr},
  {"mype",        CkLua_mype},
  {"numpes",      CkLua_numpes},
  {NULL,          NULL}
};

// construtor do chare main
main::main(CkArgMsg *m){
  // cria o estado Lua
  L = lua_open();
  // registra as bibliotecas básicas de Lua
  luaL_openlibs(L);
  luaL_register(L, "ck", CkLua_MethodsDefault);
  // executa o arquivo declarado no arquivo de interface que
  // contém a implementação do chare
  luaL_dofile(L, "main.lua");
  registerChares(L);
  // inicia a variável readonly
  mainHandle = thishandle;
  // executa a função ckNew do chare em Lua
  lua_getglobal(L, "main");
  lua_pushstring(L, "ckNew");
  lua_gettable(L, -2);
  lua_pcall(L, 0, 0, 0);
}

void main::setResult(double res) {
  // executa o método setResult da implementação de Lua,
  // passando o parâmetro res
```

```
  lua_getglobal(L, "main");
  lua_pushstring(L, "setResult");
  lua_gettable(L, -2);
  lua_pushnumber(L, res);
  lua_pcall(L, 1, 0, 0);
}

// função executada quando um entry method do chare é
// chamado de Lua
static int _main_chareIndex(lua_State * L){
  // recebe nome do método
  const char * key = lua_tostring(L, lua_upvalueindex(1));
  if (key == NULL) return 0;
  // pega o chare chamado
  CkChareID cid = *(static_cast<CkChareID*>
(lua_touserdata(L, lua_upvalueindex(2))));
  CProxy_main _c(cid);
  // executa entry method
  if (!strcmp(key, "setResult")){
    _c.setResult( (double) lua_tonumber(L, 1));
    return 0;
  }
}

// construtor do chare Calc
Calc::Calc(){
  L = lua_open();
  luaL_openlibs(L);
  luaL_register(L, "ck", CkLua_MethodsDefault);
  luaL_dofile(L, "calc.lua");
  registerChares(L);
  // registra a variável readonly do chare main
  lua_getglobal(L, "main");
  if (lua_isnil(L, -1)){
    lua_pop(L, 1);
    lua_newtable(L);
    lua_setglobal(L, "main");
    lua_getglobal(L, "main");
  }
```

```
  // registra os entry methods de main
  CkChareID* ud = (CkChareID*)
              lua_newuserdata(L, sizeof(CkChareID));
  *ud = main::mainHandle ;
  lua_newtable( L );
  lua_pushstring( L , "__index");
  lua_newtable( L );
  lua_pushstring(L, "setResult");
  lua_pushstring(L, "setResult");
  lua_pushvalue(L, -6);
  lua_pushcclosure(L, &_main_chareIndex, 2);
  lua_rawset(L, -3);
  lua_rawset( L , -3 );
  lua_setmetatable( L , -2 );
  lua_pushstring(L, "mainHandle");
  lua_pushvalue(L, -2);
  lua_settable(L, -4);
  lua_pop(L, 1);
  // executa a função ckNew da implementação do chare
  lua_getglobal(L, "Calc");
  lua_pushstring(L, "ckNew");
  lua_gettable(L, -2);
  lua_pcall(L, 0, 0, 0);
}

// chama a função putWork de Lua
void Calc::putWork(double left, double right) {
  lua_getglobal(L, "Calc");
  lua_pushstring(L, "putWork");
  lua_gettable(L, -2);
  lua_pushnumber(L, left);
  lua_pushnumber(L, right);
  lua_pcall(L, 2, 0, 0);
}

// função executa a chamda do método no grupo inteiro
static int _Calc_chareIndex(lua_State * L){
  const char * key = lua_tostring(L, lua_upvalueindex(1));
  if (key == NULL) return 0;
```

```
  CkGroupID gid = *(static_cast<CkGroupID*>
                (lua_touserdata(L, lua_upvalueindex(2))));
  CProxy_Calc _c(gid);
  if (!strcmp(key, "putWork")){
    _c.putWork( (double) lua_tonumber(L, 1),
                (double) lua_tonumber(L, 2));
    return 0;
  }
}

// função executa a chamda do método em um index do chare
static int _index_Calc_chareIndex(lua_State * L){
  const int index = (int) lua_tointeger(L, lua_upvalueindex(1));
  const char * key = lua_tostring(L, lua_upvalueindex(2));
  if (key == NULL) return 0;
  CkGroupID gid = *(static_cast<CkGroupID*>
                    (lua_touserdata(L, lua_upvalueindex(3))));
  CProxy_Calc _c(gid);
  if (!strcmp(key, "putWork")){
    _c[index].putWork( (double) lua_tonumber(L, 1),
                (double) lua_tonumber(L, 2));
    return 0;
  }
}

// função retornada quando o grupo é indexado numericamente
static int _preindex_Calc_chareIndex(lua_State * L){
  lua_pushvalue(L, lua_upvalueindex(1));
  lua_pushvalue(L, 2);
  lua_pushvalue(L, lua_upvalueindex(2));
  lua_pushcclosure(L, &_index_Calc_chareIndex, 3);
  return 1;
}

// método chamado quando o chare é acessado
static int _group_Calc_chareIndex(lua_State * L){
  // se o acesso é numérico(ex. calc[1].putWork())
  if (lua_isnumber(L, 2)) {
    lua_newtable(L);
```

```
    lua_newtable(L);
    lua_pushstring(L, "__index");
    lua_pushvalue(L, 2);
    lua_pushvalue(L, 1);
    lua_pushcclosure(L, &_preindex_Calc_chareIndex, 2);
    lua_rawset(L, -3);
    lua_setmetatable(L, -2);
    return 1;
  }
  // se o acesso é por string(ex. calc.putWork())
  else if (lua_isstring(L, 2)){
    lua_pushvalue(L, 2);
    lua_pushvalue(L, 1);
    lua_pushcclosure(L, &_Calc_chareIndex, 2);
    return 1;
  }
}

// implementação da função responsável pela instanciação
// do chare Calc através de Lua
int _Lua_static_Calc(lua_State* L){
  CkGroupID _c = CProxy_Calc::ckNew();
  CkGroupID* ud = (CkGroupID*)
              lua_newuserdata(L, sizeof(CkGroupID));
  *ud = _c;
  lua_newtable( L );
  lua_pushstring( L , "__index");
  lua_pushcfunction( L , &_group_Calc_chareIndex );
  lua_rawset( L , -3 );
  lua_setmetatable( L , -2 );
  return 1;
}
```