JULIA语言与并行计算

张常有
changyou@iscas.ac.cn
张先轶
traits.zhang@gmail.com

中国科学院软件研究所并行实验室

Julia是一个新的开源的高级编程语言，提供丰富的数据类型和精度，高效支持外部函数调用和分布式并行运行。

大纲

- 认识Julia
- 外部函数调用
- 并行计算
- 云服务平台

认识Julia

- Hello world！

```
println(“Hello world！”)
```

- 免费开源的编程语言
- MIT License（2009-）
- Library-Friendly：C/Fortran/…shared libraries
- http://julialang.org/downloads/

两种运行模式

- 两种运行模式
  - 交互式

```
D:\doing\JuliaPro>julia
               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "help()" to list help topics
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.1.2
 _/ |\__'_|_|_|\__'_|  |  c4b3649-WINNT-i686 (2013-03-08 11:45:11)
|__/                   |

julia> println("Hello world!")
Hello world!

julia>
```

  - 命令行

```
D:\doing\JuliaPro>julia helloworld.jl
Hello world!

D:\doing\JuliaPro>
```

基本程序结构

- 顺序结构

```
A = rand(5,5)
B = rand(5,5)
C = A + B
println( “C=” , C )
```

```
d:\doing\JuliaPro>julia abc.jl
C=
5x5 Float64 Array:
 0.5153   1.35792   0.919092  1.09195   1.61303
 1.06616  1.0687    1.73046   0.495719  1.0192
 1.13779  0.931572  0.714584  0.868664  1.21445
 1.37703  1.18236   0.912171  0.359207  1.31623
 1.15575  1.03046   1.00354   0.230082  1.3082
Hello world!

d:\doing\JuliaPro>
```

基本程序结构

- 选择结构

```
x=1; y=2
if x < y
    println("x is less than y")
elseif x > y
    println("x is greater than y")
else
    println("x is equal to y")
end
```

```
d:\doing\JuliaPro>julia test.jl
x is less than y
```

基本程序结构

- 循环结构

```
i=0
while i <=5 print(i+=1) end
println(";")
for i = 1:5  print(i) end;println(";")
for i in [1,4,0] print(i) end;println(";")
```

```
d:\doing\JuliaPro>julia test.jl
123456;
12345;
140;
foo
bar
baz
```

预定义的整型和浮点型

Type	Signed?	Bits
Int8	✓	8
Uint8		8
Int16	✓	16
Uint16		16
Int32	✓	32
Uint32		32
Int64	✓	64
Uint64		64
Int128	✓	128
Uint128		128
Bool	N/A	8
Char	N/A	32

- Julia 中，类型被省略，则可以是任意（Any）类型。主动添加类型说明会显著提高**性能**和系统**稳定性**。

Type	Precision	Bits
Float32	single	32
Float64	double	64

其他类型

- 有理数
- 复数
- 字符串

```
a = [1//3,3//5,7//9,8//9,2//7,9]
for i in a
   println( "double ai - ", 2i )
end
```

```
ca = 1+2im; cb = 2
println(ca + cb)
```

```
s = "Julia, a pretty language!"
println(s)
println( s[1:5],s[endof(s)])
```

复合类型

- 允许类型定义带参数

```
type Point{T}
    x::T
    y::T
end
p1=Point{Int32}(123, 456)
p2=Point{Int32}(456, 123)
println(p1);println(p2)
println("p1+p2=($(p1.x+p2.x),$(p1.y+p2.y)) " )
```

```
d:\doing\JuliaPro>julia test.jl
Point{Int32}(123,456)
Point{Int32}(456,123)
p1+p2=(579,579)
```

Julia强调运行效率

	Fortran GCC 4.5.1	Julia 12b1d5a7	Python 2.7.3	Matlab R2011a	Octave 3.4	R 2.14.2	JavaScript V8 3.6.6.11
fib	0.28	1.97	46.03	1587.03	2748.74	275.63	2.09
parse_int	9.22	1.72	25.29	846.67	7364.87	353.48	2.55
quicksort	1.65	1.37	69.20	133.46	3341.94	708.76	4.95
mandel	0.76	1.45	34.88	74.61	988.74	184.71	7.62
pi_sum	1.00	1.00	33.64	1.46	457.26	253.45	1.12
rand_mat_stat	2.23	1.95	29.01	7.71	31.04	12.66	5.53
rand_mat_mul	1.14	1.00	1.75	1.08	1.93	9.58	45.82

Figure: benchmark times relative to C (smaller is better).

大纲

- 认识Julia
- 函数调用
- 并行计算
- 云服务平台

函数

```
function sum(array)
    s = 0
    for i in array
        s += i
    end
    s
end
c = [1,3,7,8,2,4,3]
println(c)
println("sum: ",sum(c))
f = +; println(f(c[1],c[2]))
```

```
d:\doing\JuliaPro>julia test.jl
[1, 3, 7, 8, 2, 4, 3]
sum: 28
4
```

调用C/FORTRAN语言函数库

- 工作内容
  - 制作或购买共享的动态库文件
    - Windows版
    - Linux版
  - 调用过程
    - 获得库句柄（dlopen）
    - 查找需要的函数（dlsym(libc, :sum)）
    - 传递参数
    - 获得结果（ccall）
- 其他外部库
  - FFTW，BLAS

C语言库的例子-C程序

- //C语言源码文件名（testdll4julia.c）

```
#include "stdio.h"
__declspec(dllexport) int sum(int a, int b)  {
      return a + b;
}
```

- gcc -shared -o testdll4julia.dll testdll4julia.c

C语言库的例子-Julia程序


```
//Julia语言源码文件名（testclib.jl）
libc = dlopen("testdll4julia")
rs=ccall(dlsym(libc,
   :sum),Int32,(Int32,Int32),6,7)
println("sum of the two number is ",rs)
```

```
d:\doing\JuliaPro>julia testclib.jl
sum of the two number is13

d:\doing\JuliaPro>
```

用于统计的内置函数

- mean(*v* [,*region*])
- std(*v* [, *region*])
- stdm(*v*, *m*)
- var(*v* [, *region*])
- varm(*v*, *m*)
- median(*v*)
- hist(*v* [, *n*]) → e, counts
- hist(*v*, *e*) → e, counts
- histrange(*v*, *n*)
- midpoints(*e*)
- quantile(*v*, *p*)
- quantile(*v*)
- cov(v_1[, *v2*])
- cor(v_1[, *v2*])

大纲

- 认识Julia
- 函数调用
- 并行计算
- 云服务平台

对并行计算的支持

- Julia可以管理多个CPU和CPU中的多计算核心
- CPU之间的通信使用内部消息机制
- 两个基础原语
  - Remote references：an object refer to an other object
  - Remote calls: return a remote reference

- 常用函数
  - remote_call()
  - fetch()
  - remote_call_fetch()
  - nprocs()
  - addprocs()
  - addprocs _local()

- 常用宏
  - @spawn
  - @spawnat
  - @parallel

- http://docs.julialang.org

并行计算程序实例

- 问题描述：
  - 北京市现有公交线路共约3723条（上下行对开各算1条）。需要根据一整天（24小时）的运行数据，进行统计分析。
- 目标：计算每条线路各站间平均走行时长
- 初始数据：每条线路各车次到达各站的时刻

数据结构设计

- 三维数组
- x，y，z
  - x-线路，取4000
  - y-站点，取40
  - z-发车，取200

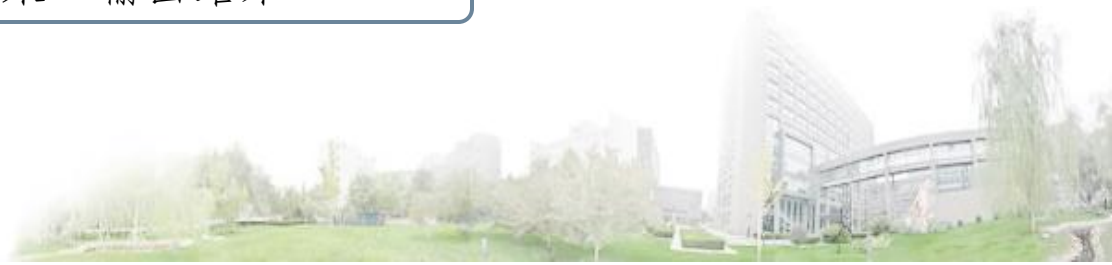
计算流程

开始：读入数据

计算站间走行时长

计算站间走行时长的平均值

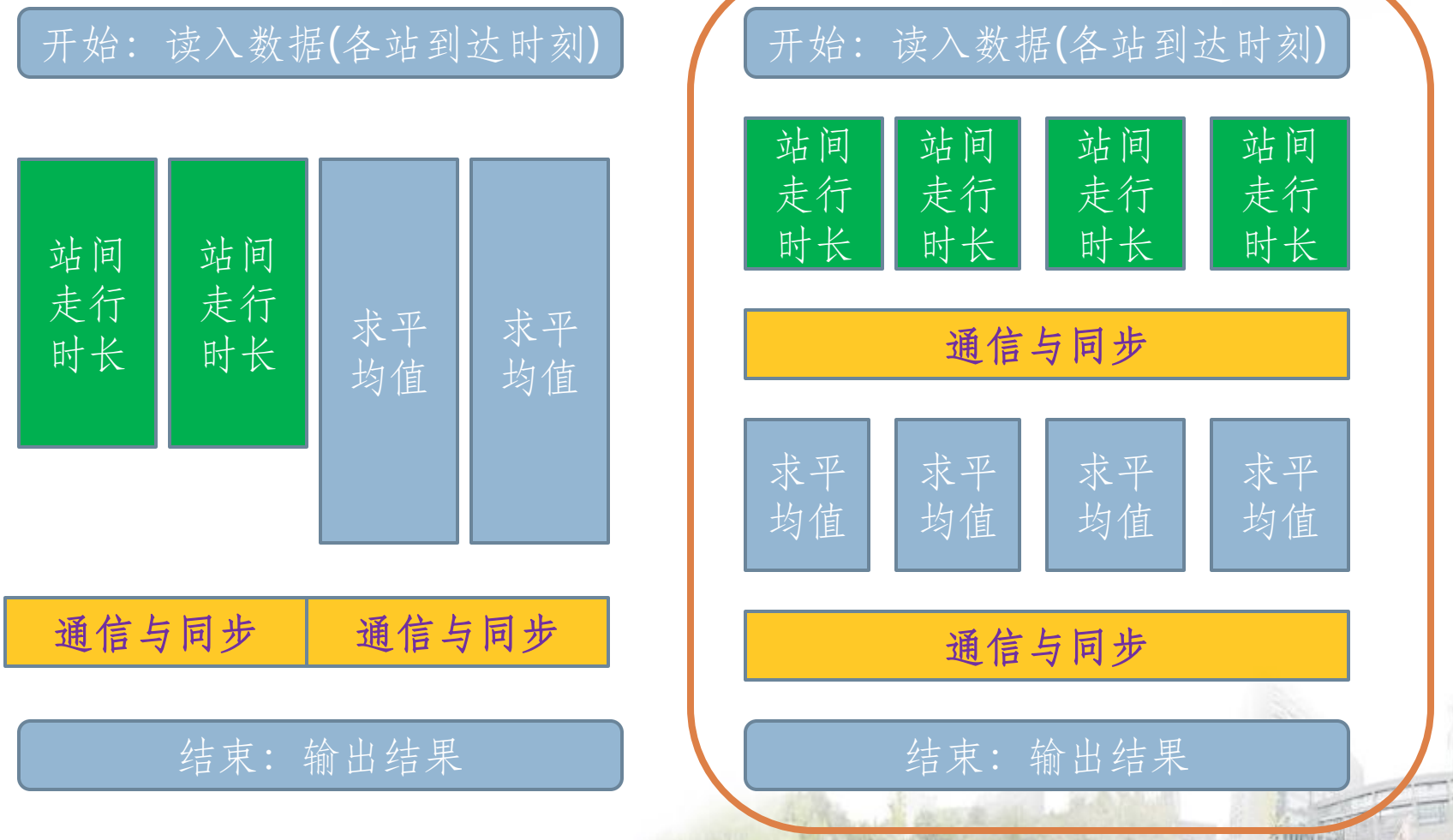
结束：输出结果



单核串行程序

- busMeanZ = Array(Float32,X,Y)
- for i = 1:X
  - for j = 1:Y
    - for k = 1:Z
      - busMeanZ[i,j] += busInter[i,j,k]
    - end
  - end
- end
- busMeanZ = busMeanZ/Z

并行策略


开始：读入数据(各站到达时刻)
站间走行时长
站间走行时长
求平均值
求平均值
通信与同步
通信与同步
结束：输出结果
开始：读入数据(各站到达时刻)
站间走行时长
站间走行时长
站间走行时长
站间走行时长
通信与同步
求平均值
求平均值
求平均值
求平均值
通信与同步
结束：输出结果

Julia并行的实现步骤

添加处理器（核心）

数据划分与传输

开始：读入数据(各站到达时刻)

站间走行时长 | 站间走行时长 | 站间走行时长 | 站间走行时长

通信与同步

求平均值 | 求平均值 | 求平均值 | 求平均值

通信与同步

结束：输出结果

julia并行程序

初始化问题参数

```
const X=4000;const Y=40; const Z=200
const NP=4; XX=1000
```

添加处理器（核心）

```
addprocs_local(3)
```

数据划分

```
pbIni1=busIni[1:XX,1:Y,1:Z]
pbIni2=busIni[XX+1:2XX,1:Y,1:Z]
pbIni3=busIni[2XX+1:3XX,1:Y,1:Z]
pbIni4=busIni[3XX+1:X,1:Y,1:Z]
```

julia并行程序

函数定义通知各核心

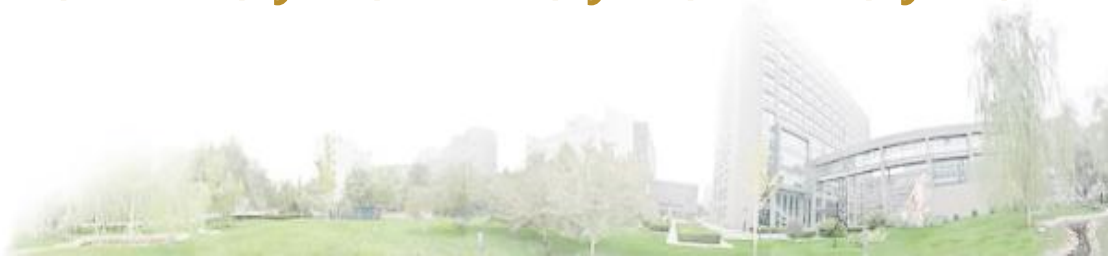
```
require("ys.jl")
```

远程调用（并行）

```
rys1=remote_call(1,ys,pbIni1,XX,Y,Z)
rys2=remote_call(2,ys,pbIni2,XX,Y,Z)
rys3=remote_call(3,ys,pbIni3,XX,Y,Z)
rys4=remote_call(4,ys,pbIni4,XX,Y,Z)
```

同步

```
wait(rys1);wait(rys2);wait(rys3);wait(rys4)
```



julia并行程序

获取各核分结果

```
pbInter1=fetch(rys1);
pbInter2=fetch(rys2);
pbInter3=fetch(rys3);
pbInter4=fetch(rys4);
```

拼合结果（时长）

```
pbInter=vcat(pbInter1,pbInter2,pbInter3,
pbInter4)
```

julia并行程序

函数定义通知各核心

```
require("zmmean.jl")
```

远程调用（并行）

```
rzm1=remote_call(1,zm,pbInter1,XX,Y,Z)
rzm2=remote_call(2,zm,pbInter2,XX,Y,Z)
rzm3=remote_call(3,zm,pbInter3,XX,Y,Z)
rzm4=remote_call(4,zm,pbInter4,XX,Y,Z)
```

同步

```
wait(rzm1);wait(rzm2);
wait(rzm3);wait(rzm4);
```

拼合结果（平均值）

```
pbMean=vcat(fetch(rzm1),fetch(rzm2),
        fetch(rzm3),fetch(rzm4))
```

实验环境

- CPU： AMD Phenom(羿龙) II X4 960T 3.0GHz - 4核
- 内存： 4 GB (宇瞻 DDR3 1333MHz)
- OS： ubuntu-12.04.2-desktop-i386
- Julia：0.2.0

计算耗时（串行）

- processor number: 1
- =计算公交线路站间行走时长(sub in Y)耗时=
- elapsed time: 11.666052409 seconds
- =计算路段平均行走时长(mean in Z)耗时=
- elapsed time: 11.810904218 seconds
- ----------
- 合计：23.58s

计算耗时（并行）

- 增加核心耗时：2.701658239 seconds
- processor number: 4
- 数据划分耗时：0.340269018 seconds
- 走行时间耗时：2.015043338 seconds
- 平均时间耗时：2.4313143 seconds
- ------------------------------
- 合计：7.49s
- ====finished====

加速比

算法类型	添加核心	数据划分	走行时长	时长平均	合计（s）	加速比
串行	--	--	11.67	11.81	23.58	3.15x
并行	2.70	0.34	2.02	2.43	7.49	

mean
优化

mean() 函数

- **DArray类型**
- **CUDA 动态库**

mean()优化

拼合结果（平均值）


```
function zm(a::Array,x::Int32,y::Int32,z::Int32)
        b=reshape(a,x,y,z)
        zmmean=Array(Float32,x,y)
        for i=1:x
                for j=1:y
                        #for k=1:z sum[i,j] += b[i,j,k] end
                        zmmean[i,j]=mean(a[i,j,1:z])
                end
        end
        #sum/z
        zmmean
end
```

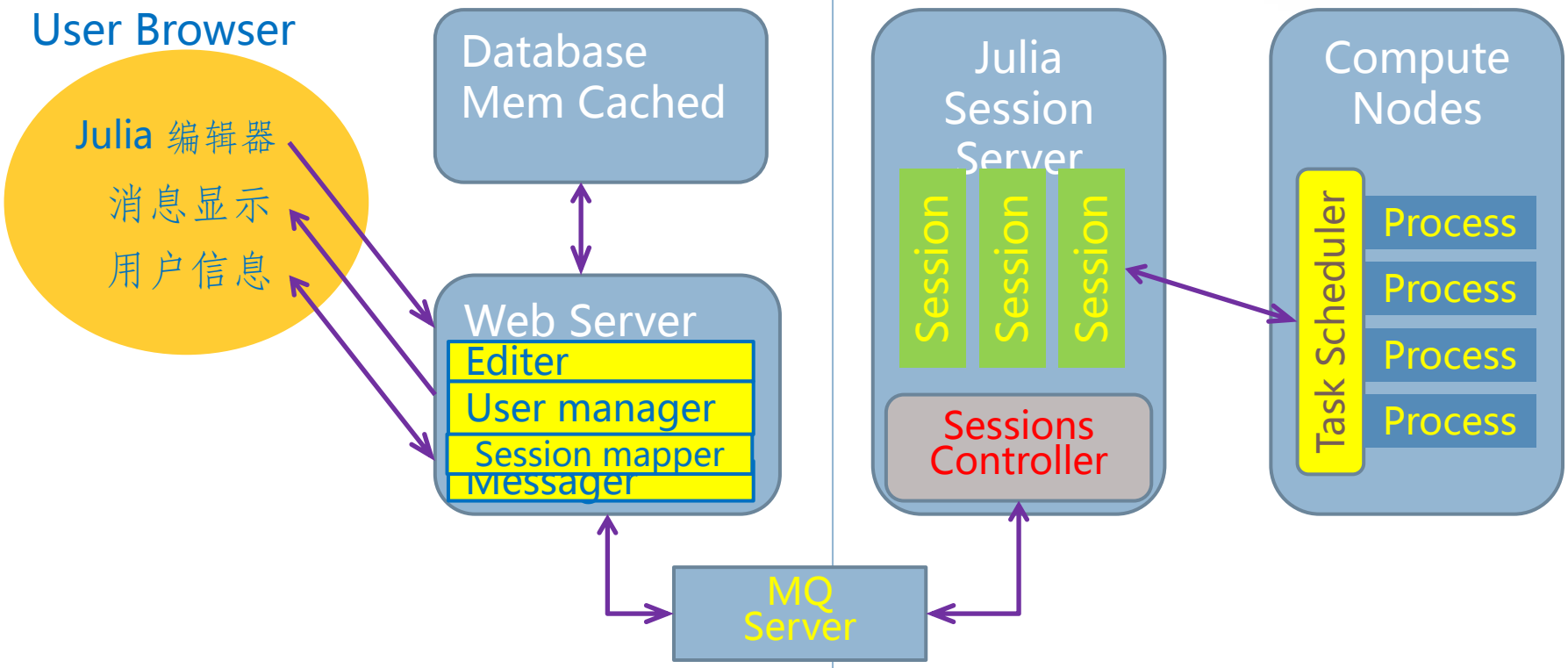
mean()函数优化

算法类型	添加核心	数据划分	走行时长	时长平均	合计（s）	加速比
串行	--	--	11.67	11.81	23.58	3.15x
并行	2.70	0.34	2.02	2.43	7.49	
mean优化	--	--	11.60	1.67	13.27	1.91x
	2.63	0.20	2.07	2.04	6.94	

大纲

- 认识Julia
- 函数调用
- 并行计算
- 云服务平台

云服务平台



目标：

轻松编写和调试Julia程序，驾驭超级计算机，开发或使用高性能软件（库），拓展计算能力，高效解决应用问题。

OpenBLAS：开源高性能BLAS库

- 张先轶
- traits.zhang@gmail.com

为什么做OpenBLAS

- 各大CPU厂商都有商业数学库
  - Intel MKL，AMD ACML，IBM ESSL
- 开源实现
  - ATLAS
    - 自适应优化技术，性能一般
  - GotoBLAS
    - 手工汇编优化，最优实现
    - 开发者Kazushige Goto 2010年离
    - 已停滞
- 2011年初发起OpenBLAS
  - 基于GotoBLAS2 1.13 BSD版

OpenBLAS简介

- 目标：成为全球最好的BLAS开源实现
- BSD协议, 当前稳定版本 0.2.6
- 当前开发人员
  - 张先轶, 王茜, Zaheer Chothia
- 主要进展
  - 完成龙芯3A 处理器支持和优化
  - 完成Intel Sandy Bridge BLAS 3级优化
  - AMD Bulldozer S/DGEMM改进
- 细节改进
  - 增强Mac OS X, Windows, FreeBSD上的编译、安装和使用
  - 修正各种bug: SEGFAULT,计算结果错等等

OpenBLAS性能结果

- 龙芯3A CPU
  - BLAS3级 4线程，OpenBLAS超过GotoBLAS 120%，ATLAS 73%

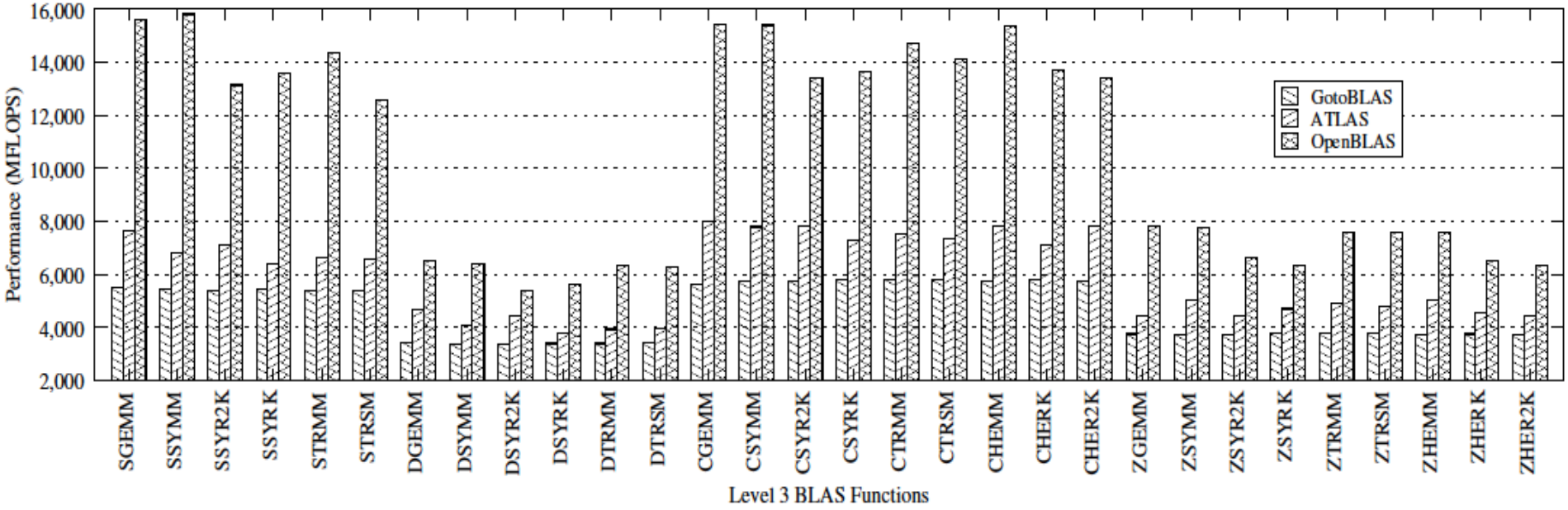
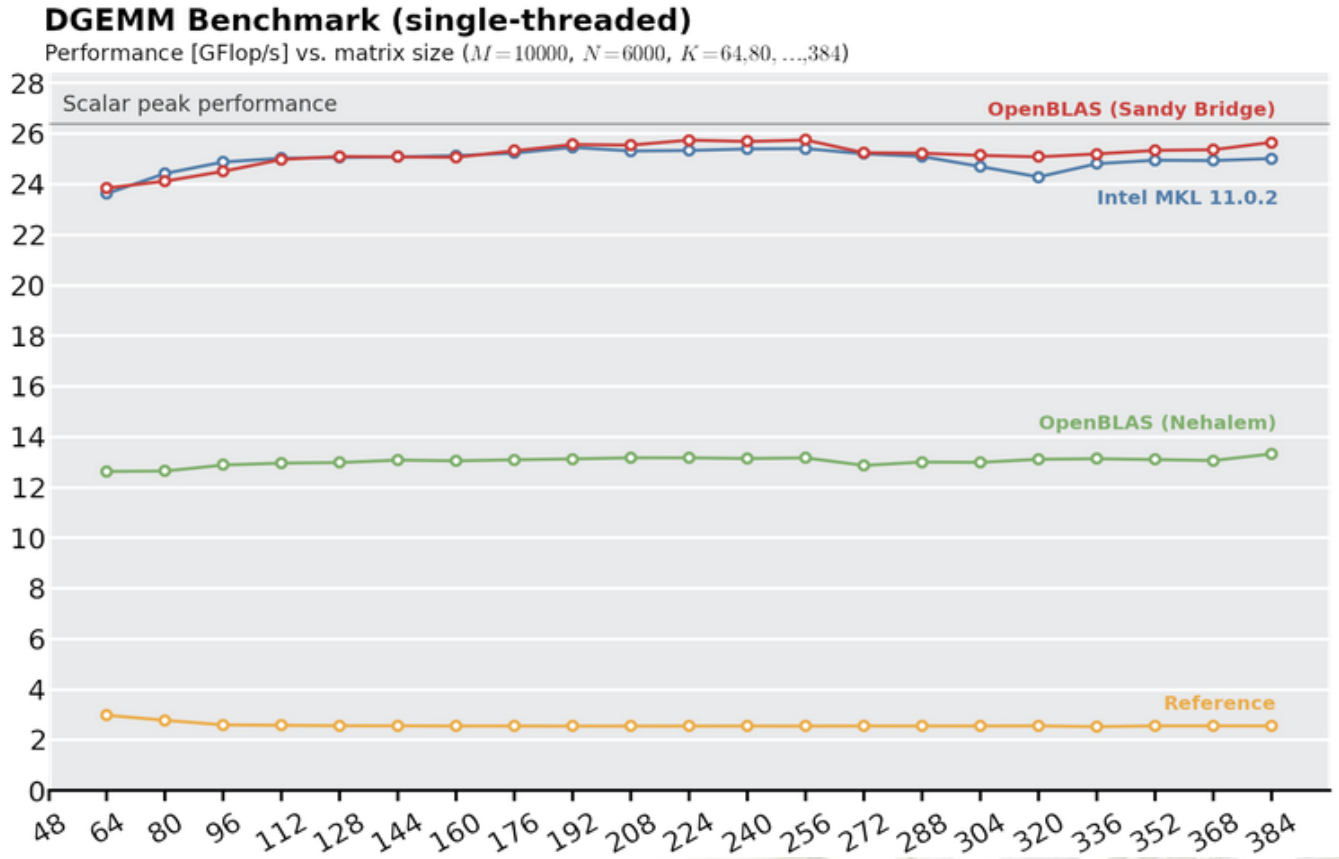


Figure 11. Multi-threaded Level 3 BLAS Performance (NP=4)

Zhang Xianyi, Wang Qian, Zhang Yunquan, Model-driven Level 3 BLAS Performance Optimization on Loongson 3A Processor, 2012 IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 17-19 Dec. 2012

OpenBLAS性能结果

- Intel Sandy Bridge CPU
  - 与Intel MKL相当



DGEMM performance on Intel Core i5-2500K (Sandy Bridge) , Windows 7 SP1

总结

- Julia易用
- 效率高
- 支持并行计算
- 支持C/Fortran共享动态库
- 提供丰富的统计函数（优化）
- 我们的工作
  - 高性能云服务平台
  - OpenBlas

欢迎各位的使用和反馈

谢谢！

主页：https://github.com/xianyi/OpenBLAS

邮件列表：https://groups.google.com/forum/#!forum/openblas-users

Issue：https://github.com/xianyi/OpenBLAS/issues