Go’s Concurrency Constructs on the SCC

Andreas Prell
Thomas Rauber

MARC Symposium 2012 - ONERA, Toulouse
Goroutines
Channels

Photo: 48-core SCC processor, Intel
Go is ...

• A general-purpose programming language
• Statically typed
• Garbage collected
• Object-oriented, but without type hierarchy
Go supports concurrent programming

- Goroutines (think lightweight threads)
- Channels for communication and synchronization
- Concurrency model inspired by CSP

Go-style concurrency on the SCC?

**Go:** “Don’t communicate by sharing memory. Instead, share memory by communicating.”

**SCC:** “In SCC we imagined messaging instead of shared memory or shared memory access coordinated by messages. [...] Use a message to synchronize, not a memory location.” *

* Jim Held, Intel
http://communities.intel.com/message/113676#113676
http://communities.intel.com/message/115657#115657
Intel Single-Chip Cloud Computer

48-core “cluster-on-a-chip”

Software research platform

6x4 tile array

2-core tile
Intel Single-Chip Cloud Computer

48-core “cluster-on-a-chip”

Software research platform

6x4 tile array

2-core tile

16 KB Message Passing Buffer
Goroutines

Goroutines hide many of the complexities of thread creation and management:

\texttt{go} \ f(x) \ starts \ f(x) \ as \ a \ \texttt{goroutine}

\texttt{f(x)} \ runs \ \texttt{concurrently}
Goroutines as Tasks on the SCC

- `go f(x)` creates a task to run `f`
- Tasks are scheduled by work-stealing (on-chip)
- Worker threads run tasks as coroutines
- Tasks that block can yield control
Simple example

```c
void sum(int x, int y) {
    printf("%d + %d = %d\n", x, y, x + y);
}

go sum(1, 2);
sleep(1);
puts("That’s it");
```

After compiling and linking with libgo:

```
$ run -n2 ./a.out
Running “a.out” on 2 cores ...
  rck01: 1 + 2 = 3
  rck00: That’s it
```
What happens with the go statement?

// Data structure to hold the goroutine’s arguments
typedef struct { int x; int y; } sum_task_data;

// Gets called by the runtime, wraps call to sum
void sum_task_func(sum_task_data *data)
{
    int x = data->x;
    int y = data->y;
    sum(x, y);
}

// Creates and enqueues a sum task
void go_sum(int x, int y)
{
    Task task;
    sum_task_data *data = (sum_task_data *)task.data;
    task.fn = (void (*)(void *))sum_task_func;
    data->x = x;
    data->y = y;

    // Enqueue task
}

go_sum(1, 2);
sleep(1);
puts("That’s it");
Permit goroutines to communicate by sending and receiving values

// An unbuffered (synchronous) channel for ints
Channel *ch = channel_alloc(sizeof(int), 0);

// Blocks until n has been received
channel_send(ch, &n, sizeof(int));

On the SCC: ring buffer in on-chip memory
No collective allocations!

```c
void f(Channel *ch);

typedef struct { int ch_own; int ch_off; } f_task_data;

void f_task_func(f_task_data *d)
{
    Channel *ch = (Channel *)(RCCE_comm_buffer[d->ch_own] + d->ch_off);
    f(ch);
}
```

Core i:

```c
b = RCCE_malloc(...);
o = b - RCCE_comm_buffer[i];
// send o to core j
```

Core j:

```c
// receive o from core i
p = RCCE_comm_buffer[i] + o;
// put/get using p
```
Simple example with coordination

```c
void sum(int x, int y, Channel *ch)
{
    int s = x + y;
    printf("%d + %d = %d\n", x, y, s);
    channel_send(ch, &s, sizeof(s));
}

Channel *ch = channel_alloc(sizeof(int), 0);
go sum(1, 2, ch);
channel_receive(ch, &tmp, sizeof(tmp));
puts("That’s it");
```

After compiling and linking with libgo:

```
$ run -n2 ./a.out
Running “a.out” on 2 cores
...
  rck01: 1 + 2 = 3
  rck00: That’s it
```
Another example

```c
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;

    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
```
Another example

```c
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
```
Another example

```c
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;)
    {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++)
    {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
```

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...
Another example

```c
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
```
Another example

```c
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
```
void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;)
    {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    go filter(ch, ch1, prime);
    ch = ch1;
}
Another example

void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;)
    {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++)
    {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
void generate(Channel *ch) {
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime) {
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n) {
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}
Another example

void generate(Channel *ch)
{
    int i;
    for (i = 2;; i++)
        channel_send(ch, &i, sizeof(i));
}

void filter(Channel *in, Channel *out, int prime)
{
    int i;
    for (;;) {
        channel_receive(in, &i, sizeof(i));
        if (i % prime != 0)
            channel_send(out, &i, sizeof(i));
    }
}

void prime_sieve(int n)
{
    Channel *ch = channel_alloc(sizeof(int), 0);
    int prime, i;
    go generate(ch);
    for (i = 0; i < n; i++) {
        Channel *ch1 = channel_alloc(sizeof(int), 0);
        channel_receive(ch, &prime, sizeof(prime));
        printf("%d\n", prime);
        go filter(ch, ch1, prime);
        ch = ch1;
    }
}

Works! But without garbage collection... not that useful
Channels - Latency

32-byte ping-pong

- RCCE send/recv
- Channel send/recv

Round-trip latency (usecs)

Number of on-chip network hops

533 MHz tile, 800 MHz mesh, 800 MHz DDR3
Parallel Genetic Algorithm

- Selection, crossover, mutation
- Concurrent evolution on $n$ islands
- Islands are created with `go evolve(...)`

![Diagram showing the connection between islands](image)
Parallel Genetic Algorithm

- Migration between neighboring islands
- Individuals are exchanged over channels

Diagram:

- Circles represent islands:
  - Island 0
  - Island 1
  - Island 2
  - Island 3

- Arrows indicate the direction of migration:
  - Send from Island 0 to Island 1
  - Receive by Island 1
  - Send by Island 2
  - Receive by Island 1
  - Send by Island 3
  - Receive by Island 2

Send and Receive labels illustrate the exchange process.
Parallel Genetic Algorithm

• Toy problem: evolve a simple “Hello World!” program from random garbage

• GA is overkill, but anyway...

```c
#include <stdio.h>
int main(void)
{
    printf("Hello SCC!\n");
    return 0;
}
```

Example: fitness = 5022

```
“mnc{pot17us\`jy-u6(
 euw!ofen#yqsl%$
x*
zvosqo&!8`fbb OAX*Nb%*T&
 uui{rs+<A”
 v

Example: fitness = 5022

fitness = \sum_{i=0..n} (target[i] - src[i])^2
```
Parallel Genetic Algorithm

Total population size: 1280

Migration rate: two individuals every ten generations

533 MHz tile, 800 MHz mesh, 800 MHz DDR3
Parallel Genetic Algorithm

Total population size: 1280

Migration rate: two individuals every ten generations

533 MHz tile, 800 MHz mesh, 800 MHz DDR3
Parallel Genetic Algorithm

Total population size: 1280

Migration rate: two individuals every ten generations

RCCE version can’t handle more islands than cores!

533 MHz tile, 800 MHz mesh, 800 MHz DDR3
Parallel Genetic Algorithm

Total population size: 1280

Without migration

533 MHz tile, 800 MHz mesh, 800 MHz DDR3
Summary

• Goroutine implementation on top of tasks
• Work-stealing of goroutines
• Channels allocated in on-chip memory
• Not addressed here: select control structure, importance of garbage collection