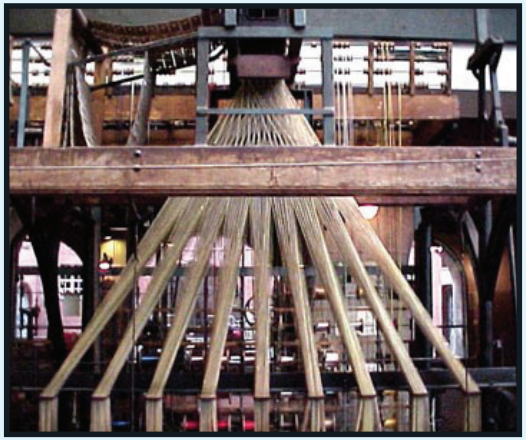
The Stitch Programing Language

-or-

How I Learned To Stop Worrying and Love P-Threads.

Motivation

Most "modern" programming languages trace their origins back decades to before the advent of cheap, general purpose multicore CPUs. They were designed for a distinctly mono-threaded environment. With Stitch, we aimed to build a language that has the power and flexibility of a fully compiled C style language, while having native threading support for modern multi-threaded applications.



Same old C

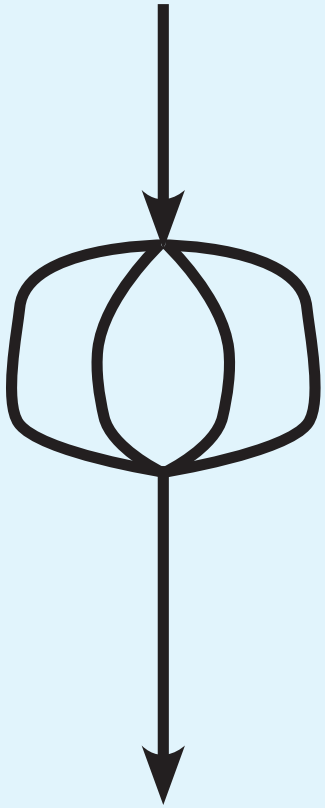
```
int gcd(int a, int b) {
        while (a != b) {
                if (a > b) {
                        a = a - b;
                }
                else {
                        b = b - a;
                }
        }
        return a;
}
```

Easy(er) Multi-threading

```
stitch i from 0 to 255 by 1: {
      acc = a[i] * b[i];
}

print(acc)
```

Easy(er) Multi-threading

```
int i = 0;
if (a[i] != b[i]){

      stitch i from 0 to 255 by 1: {
            acc = a[i] * b[i];
      }

      print(acc)
}
```

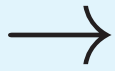


Under The Hood

- Wrap everything in pthreads
- Body of the Stitch loop becomes a function
- Pointer to the function is passed into the pthread
- Struct with copies of all variables in the scope of the Stitch loop passed in.
- (Very) limited variety of accumulators can be used in Stitch loops, and are then reconciled automatically.

Not Very Pretty

```
int main() {

        int a[255] = {1,..., 1};
        int b[255] = {2,..., 2};
        int_am acc;
        int i = 0;

        stitch i from 0 to 255 by 1: {
                acc = a[i] * b[i];
        }

        print(acc);

        return 0;
}
```



(But it works)

```
#include ‘‘stch_headers.h’’

struct stch_rangeInfo_0 {
int begin;
int end;
int stepSize;
int i;
int acc;
int b;
int a;
};

void *_0 (void *vars){{
((struct stch_rangeInfo_0 *)vars)->acc = a[i] * b[i];
}

return (void*)0;
}

int main()
{
int a[255] = {1,..., 1};
int b[255] = {2,..., 2};
int acc;
int i = 0;

pthread_t *threadpool_0 = malloc(NUMTHREADS * sizeof(pthread_t));
struct stch_rangeInfo_0 *info_0 = malloc(sizeof(struct stch_rangeInfo_0)
* NUMTHREADS);
int thread_0 = 0;
for(i = 0;i < 255;i = i+255/NUMTHREADS) {
info_0[thread_0].begin = i;
info_0[thread_0].i = i;
info_0[thread_0].acc = acc;
info_0[thread_0].b = b;
info_0[thread_0].a = a;

if((i + 2*(255/NUMTHREADS)) > 255) {
info_0[thread_0].end = 255;
i = 255;
}
else {
info_0[thread_0].end = i + 255/NUMTHREADS;
}
int e = pthread_create(&threadpool_0[thread_0], NULL, _0, &info_0[thread_0]);
if (e != 0) {
perror(‘‘Cannot create thread!’’);
free(threadpool_0); //error, free the threadpool
exit(1);
}
thread_0++;
}

//loop and wait for all the threads to finish
for(i = 0; i < NUMTHREADS; i++) {
pthread_join(threadpool_0[i], NULL);
}
printf(‘‘%d\n’’, acc);
return 0;
}
```

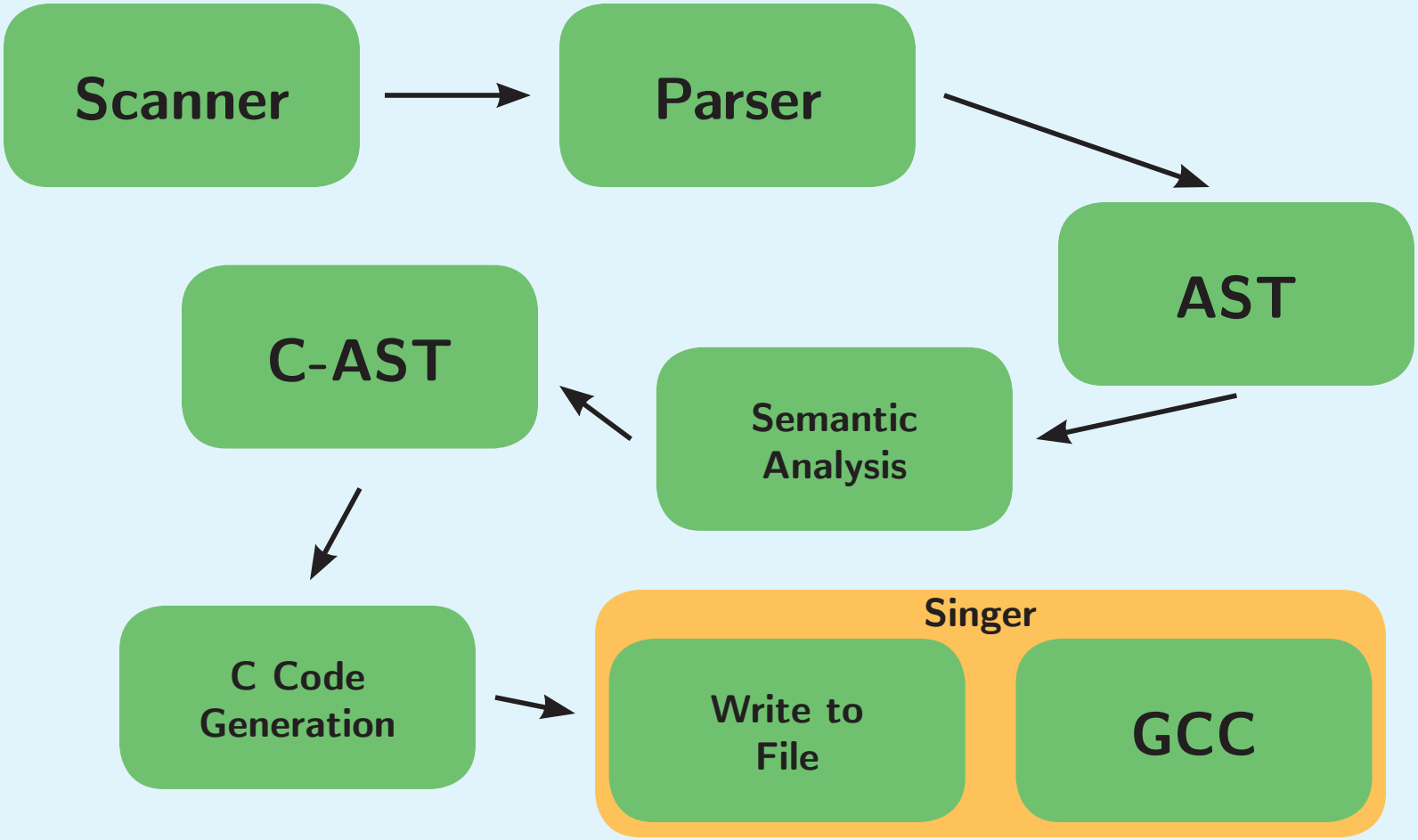
Pros

- Much simpler to write
- No dirty mutex's
- Automatic splitting of workload

Cons

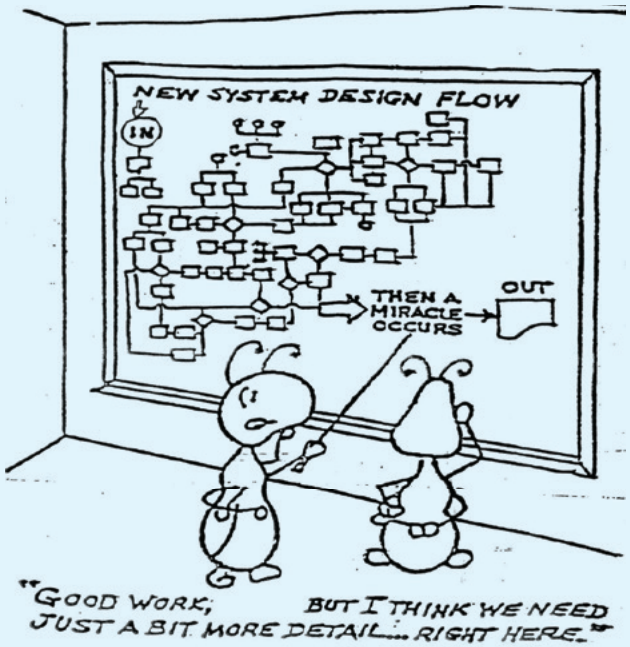
- Limited in application
- Nested Stitch loops give no benefit, but add overhead
- Pthread code definitely not optimized (but not too bad either)

And now for something completely the same...



Highpoints of C Code Generation

- Stitch to pthread loop.
- Body of the Stitch loop is turned into a C function.



- Stitch functions and range info are named procedurally.
- print() and error() are dynamically typed into proper printf() call.

Testing

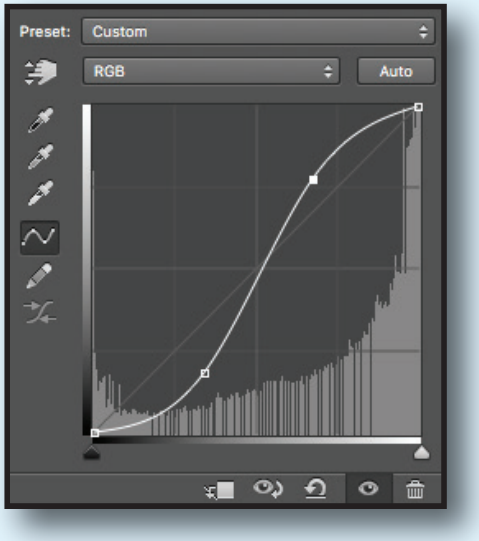
- Automated test suite
- Generate C Code and compile.
- Run compiled C, diff the output
- Positive (should compile and generate good output).
- Negative (shouldn't compile)
- Modular tests $\rightarrow$ anyone can add

Demo

- Something parallelizeable
- Something interesting
- Something that we could see the results of

Image Curves

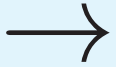
- Used all the time in Photoshop
- Map input to output based on predefined curve

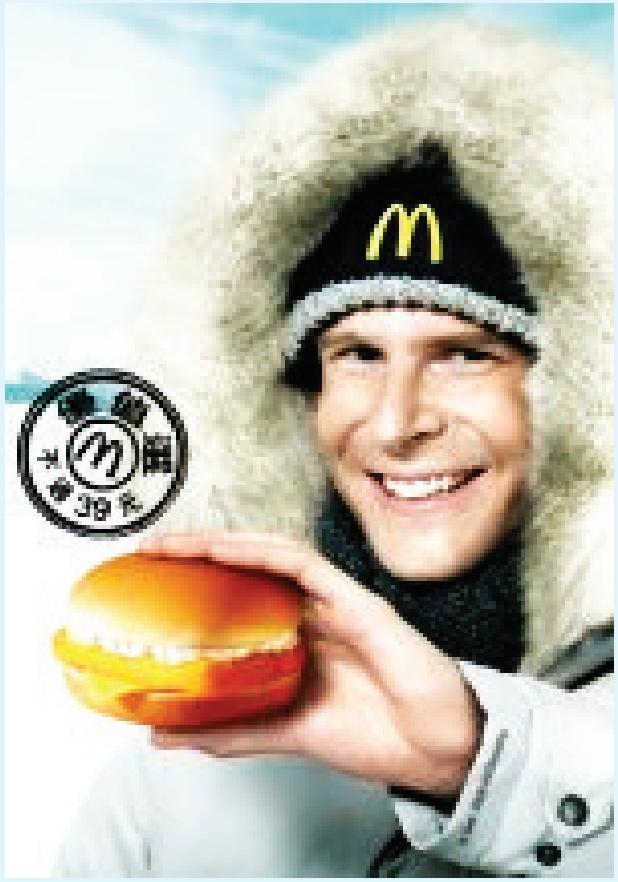


Simplest Curve - Invert

$\rightarrow$

More Useful - Increase Contrast





Code (Stitch)

```
/* Image Inverter */

int main(){

        int curve[256] = { 255, ..., 2, 0 };

        /*File IO*/

        int i = 0;
        //55 = header offset, 98592 = size of file
        stitch i from 55 to 98592 by 1:{
                int tmp = <FILE>[i];
                <FILE>[i] = curve[temp];
        }

        /*More IO*/
}
```