دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

گزارش پروژه

عنوان:

پروژه شماره ۷ ـ شبکه‌های میان‌ارتباطی

نگارش:

امیرمهدی نامجو و علیرضا فرودنیا

استاد:

دکتر حمید سربازی آزاد

زمستان ۱۴۰۲


بسم الله الرحمن الرحیم

فهرست مطالب

فهرست شکل‌ها

فصل ۱

مقدمه

شبیه‌ساز Booskim یکی از مهم‌ترین شبیه‌سازهای حضوزه شبکه‌های میان‌ارتباطی است که سالیان طولانی است در این حوزه مورد استفاده قرار گرفته و در مقالات و پروژه‌های زیادی به منظور شبیه‌سازی ایده‌های نوین این حوزه مورد استفاده قرار گرفته است. این شبیه‌ساز براساس مطالب کتاب Principles and Practices of Interconnection Networks ساخته شده است.

در این پروژه قصد داریم به شبیه‌سازی یک k-ary n-cube یا همان شبکه توری مدور (Torus) پرداخته و الگوریتم مسیریابی Duato را با حداکثر تعداد کانال‌های ممکن برای مسیریابی Fully Adap-tive و حداقل تعداد ممکن برای مسیریابی Deadlock Free پیاده‌سازی کنیم.

منابع مورد استفاده برای این گزارش [۱] و [۲] بوده‌اند.

فصل ۲

مفاهیم پایه و نحوه کارکرد شبیه‌ساز

۲ـ۱ مفاهیم پایه

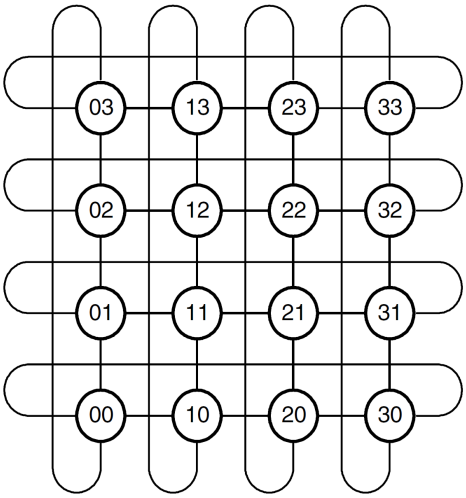
در ابتدا به طور مختصر به مفاهیم پایه مورد استفاده در این پروژه می‌پردازیم.

همان طور که در مقدمه گفته شد، قرار است با یک k-ary n-cube کار کنیم. نام دیگر این شبکه توری مدور [۱] است و شبکه‌ای است که از نظر طراحی کلی، به شدت شبیه شبکه توری عادی [۲] است با این تفاوت که رئوس ابتدا و انتهای هر بعد هم به یکدیگر متصل هستند و عملا در هر بعد یک حلقه [۳] را ایجاد می‌کنند. در این توپولوژی، n نشان‌دهنده تعداد ابعاد و k نشان‌دهنده تعداد گره در هر بعد است. باید توجه کرد که Torus به شکل کلی می‌تواند در هر بعد گره‌های متفاوتی داشته باشد ولی k-ary n-cube شکل خاصی از Torus است که همه بعدهای آن اندازه یکسانی دارد. یک نمونه از آن را در شکل ۲ـ۱ مشاهده می‌کنید.

از جمله ویژگی‌های مهم k-ary n-cube می‌توان به موارد زیر اشاره کرد:

- سایز شبکه: k^n
- درجه رئوس: n

[1] Torus
[2] Mesh
[3] Ring



شکل ۲ ـ ۱: یک 4-ary 2-cube

- قطر: $n(k-۱)$
- منظم است.
- متقارن است.
- درجه اتصال: $\frac{n}{k^{n-۱}}$
- عرض برشی: $\frac{k^n}{k} = k^{n-۱}$

مسئله دیگر، مسیریابی Duato است. به طور کلی ما علاقه‌داریم که درجه Adaptivity الگوریتم مسیریابی ما بالا باشد تا سیستم ما کارایی بالاتری داشته باشد. در شبکه‌های Fully Adaptive امکان استفاده از همه کانال‌ها وجود داشته و درجه Adaptivity برابر ۱ می‌شود. با این حال مشکلی که وجود دارد این است که ممکن است الگوریتم ما امکان بن‌بست داشته باشد. بن‌بست به معنی وابتسگی حلقوی چندین گره به هم برای عبور پیام است و در این صورت هیچ کدام امکان آن را نخواهند داشت. با این حال می‌دانیم تعدادی الگوریتم Deadlock Free هم وجود دارند که نبود بن‌بست را تضمین می‌کنند. با این وجود پرفرمنس این الگوریتم‌ها لزوما خوب نیست.

روش Duato به این صورت است که تعدادی کانال مجازی به منظور اجرای یک الگوریتم Deadlock Free قرار می‌دهد و از سایر کانال‌های مجازی پیام‌ها با یک الگوریتم Fully Adaptive بهینه عبور داده می‌شوند. در صورتی که در قسمتی از کار متوجه بن‌بست شدیم، پیام به کانال‌های مجازی که به منظور

انتقال پیام‌ها به صورت Deadlock Free تعیبه شده‌اند منتقل شده و با الگوریتم آن قسمت انتقال خود را انجام می‌دهد. به این ترتیب ما هم می‌توانیم از مزایای الگوریتم‌های سریع Fully Adaptive که لزوما تضمین برای نبود بن‌بست نمی‌دهند استفاده کنیم و هم از مزایای الگوریتم‌هایی که به ما تضمین می‌دهند بن‌بستی وجود نخواهد داشت.

۲ـ۲ نحوه کارکرد شبیه‌ساز

شبیه‌ساز Booksim شبیه‌ساز قدرت‌مندی است که بسیاری از نیازهای مربوط به شبیه‌سازی شبکه‌های میان‌ارتباطی در آن پیش‌بینی شده است. از پیاده‌سازی الگوریتم‌های مختلف مسیریابی گرفته، تا توپولوژی‌های مختلف و حتی ریزمعماری روترها. این ساختار به شکلی کاملا Modular طراحی شده و برای تغییر در یک قسمت خاص نیازی نیست که در سایر قسمت‌ها تغییر بزرگی ایجاد بشود.

علاوه بر این، شبیه‌ساز Booksim ساختار خاصی برای فایل‌های Config خود دارد و براساس ساختار این فایل‌ها اقدام به شبیه‌سازی شبکه می‌کند. در زیر نمونه‌ای کوچک از ساختار یک فایل Config آمده است.

```
// Topology
topology = torus;
k = 16;
n = 2;
// Routing
routing_function = dim_order;
// Flow control
num_vcs = 4;
// Traffic
traffic = uniform;
injection_rate = 0.3;
sim_type = throughput;
```

در ساختار بالا، topology مشخص کننده نوع توپولوژی و مقادیر k و n مشخص کننده پارامترهای آن هستند. num_vcs نشان‌دهنده تعداد کانال‌های مجازی است. traffic نشان‌دهنده نوع ترافیک ورودی، injection_rate نرخ ورود بسته‌ها و در نهایت sim_type مشخص می‌کند که پردازش‌ها متمرکز بر نرخ تاخیر باشد یا throughput.

در بین فایل‌های اصلی که در این پروژه وجود دارد، فایل‌های پوشه routers برای شبیه‌سازی Router ها بوده و فایل‌های پوشه networks تعریف‌کننده انواع توپولوژی‌هاست. همچنین فایل routerfunc مربوط به الگوریتم‌های مسیریابی مختلف است. عمده تغییرات ما مربوط به routerfunc خواهد بود.

فصل ۳

پیاده‌سازی

از آن‌جایی که خود توپولوژی Torus در Booksim پیاده‌سازی شده است، پیاده‌سازی ما معطوف به قسمت مسیریابی خواهد بود. با خواندن routerfunc ایده و شکل کلی نوشتن توابع مربوط به را بدست می‌اوریم. مهم‌ترین نکته این است که شیوه Duato، یا به بیان مورد استفاده در شبیه‌ساز Escape VC، در الگوریتم min_adaptive_torus هم به کار رفته است و می‌توانیم از آن الگو بگیریم.

به طور خلاصه، کاری که برای پیاده‌سازی Duato انجام می‌شود این است که دو کانال مجازی اول به الگوریتم Deadlock Free تعلق می‌گیرند. برای این کار از الگوریتم Dimension Order Routing که در خود Booksim پیاده‌سازی شده است استفاده می‌کنیم. سایر کانال‌های مجازی را به پیاده‌سازی الگوریتم Fully-Adaptive اختصاص می‌دهیم.

بدین منظور بخش‌های مختلف کد که در تابع fully_adaptive_torus نوشته شده‌اند را مورد بررسی قرار می‌دهیم.

```
  void fully_adaptive2_torus( const Router *r, const Flit *f,

  int in_channel, OutputSet *outputs, bool inject ){
int vcBegin = 0, vcEnd = gNumVCs-1;
if ( f->type == Flit::READ_REQUEST ) {
        vcBegin = gReadReqBeginVC;
        vcEnd = gReadReqEndVC;
```

```
} else if ( f->type == Flit::WRITE_REQUEST ) {
        vcBegin = gWriteReqBeginVC;
        vcEnd = gWriteReqEndVC;
} else if ( f->type ==  Flit::READ_REPLY ) {
        vcBegin = gReadReplyBeginVC;
        vcEnd = gReadReplyEndVC;
} else if ( f->type ==  Flit::WRITE_REPLY ) {
        vcBegin = gWriteReplyBeginVC;
        vcEnd = gWriteReplyEndVC;
}
assert(((f->vc >= vcBegin) &&
 (f->vc <= vcEnd)) || (inject && (f->vc < 0)));


outputs->Clear( );
```

این قسمت در ابتدای همه توابع مسیریابی وجود دارد و مربوط به این است که بسته به نوع فلیت، کانال‌های آن تعیین بشود. ما هم این را در کد خودمان قرار داده‌ایم ولی عملا در کاربرد ما خیلی این مورد بررسی نمی‌شود و به شکل کلی می‌توانیم در بررسی‌های خود vcBegin را معادل ۰ و vcEnd را هم معادل تعداد کانال‌ها منهای یک در نظر بگیریم. در خط آخر هم مجموعه Output خالی شده است. این مجموعه اصلی‌ترین خروجی تابع است که مشخص می‌کند خروجی‌های هر کدام از بخش‌های روتر باید از کدام کانال خارج شوند و عملا کانال و جهت خروجی را تعیین می‌کند.

```
if(inject) {
        // injection can use all VCs
        outputs->AddRange(-1, vcBegin, vcEnd);
        return;
} else if(r->GetID() == f->dest) {
```

```
        // ejection can also use all VCs
        outputs->AddRange(2*gN, vcBegin, vcEnd);
}

int in_vc;
if ( in_channel == 2*gN ) {
        in_vc = vcEnd; // ignore the injection VC
} else {
        in_vc = f->vc;
}

int cur = r->GetID( );
int dest = f->dest;

int out_port;
int dim_left[gN] = {};
int dim_left_cur[gN] = {};
int dim_left_dest[gN] = {};
int dim_left_cur2[gN] = {};
int dim_left_dest2[gN] = {};
int dim;
```

این قسمت هم در سایر توابع مربوط به Torus آورده شده است. دو if اول عملا برای مشخص کردن وضعیت Injection و Ejection از سوئیچ هستند. شرط بعدی برای حالتی است که کانال ورودی عملا آخرین کانال باشد. و در نهایت یکسری متغیر که در قسمت‌های بعدی استفاده می‌شوند تعریف شده است.

```
if ( in_vc > ( vcBegin + 1 ) ) {
```

```
        part1
}
```

این شرط، برای این است که دو کانال اول برای حالت Deadlock Free استفاده شده و سایر کانال‌ها برای الگوریتم Fully Adaptive استفاده شده‌اند.

```
//part1.1

 int dist2;
int arrived_at_dest = 1;
for (dim = 0; dim < gN; ++dim) {
        if ((cur % gK) != (dest % gK)) {
                dim_left[dim] = 1;
                dim_left_cur[dim] = cur;
                dim_left_dest[dim] = dest ;

                dim_left_cur2[dim] = cur % gK;
                dim_left_dest2[dim] = dest % gK;

        }
        cur /= gK;
        dest /= gK;
}
for (int i = 0; i < gN; i++) {
        if (dim_left[i] == 1) {
                arrived_at_dest = 0;
                break;
        }
}
```

در این قسمت در هر بعد اختلاف مقصد و نقطه فعلی را بررسی می‌کنیم. در صورتی که در جایی اختلاف داشته باشند، مشخص می‌کنیم که آن نقطه کدام است و این نقاط را در یک آرایه نگه می‌داریم. توجه کنید که دو آرایه داریم که یکی مقدار اولیه و باقی‌مانده مقدار اولیه بر k را نگه می‌دارد. این موضوع به دلیل این است که در آینده هم به باقی‌مانده آن بر k نیاز داریم و هم خودش. (این قسمت عملا تعمیم یافته الگوریتم `min_adapt` خود booksim است.)

```
//part1.2

if (arrived_at_dest == 0) {
dim = getRandomOneIndex(dim_left);
cur = dim_left_cur[dim];
dest = dim_left_dest[dim];

int cur2 = dim_left_cur2[dim];
int dest2 = dim_left_dest2[dim];

dist2 = gK - 2 * ((dest2 - cur2 + gK) % gK);

if ((dist2 > 0) ||
((dist2 == 0) && (RandomInt(1))))
{
        out_port = 2 * dim; // Right

} else {
        out_port = 2 * dim + 1; // Left

}
// printf("HELLO %d %d\n" , vcBegin, vcEnd );
outputs->AddRange(out_port, vcBegin + 3, vcEnd,1);
```

```
}
```

در این قسمت، از بین خانه‌هایی که پیدا کرده‌ایم که در آن بعد برابر نیستند، یکی را به تصادف انتخاب کرده، و سپس این که از سمت چپ یا راست Torus نزدیک‌تر است را محاسبه می‌کنیم و بر همین اساس از کانال مربوطه آن را انتقال می‌دهیم. (برای کانال راست از 2␣*␣dim و برای کانال چپ از 2␣*␣dim␣+1) استفاده می‌کنیم.

عملا کاری که این‌جا انجام شده مشابه کاری است که در کدهای از قبل آماده شده Booksim در الگوریتم min_adapt_torus انجام شده ولی به جای این که برای اولین خانه‌ای که اختلاف دارد انجام بشود، برای همه انجام می‌دهیم و سپس بین آن‌هایی که اختلاف وجود دارد یکی را به تصادف انتخاب می‌کنیم.

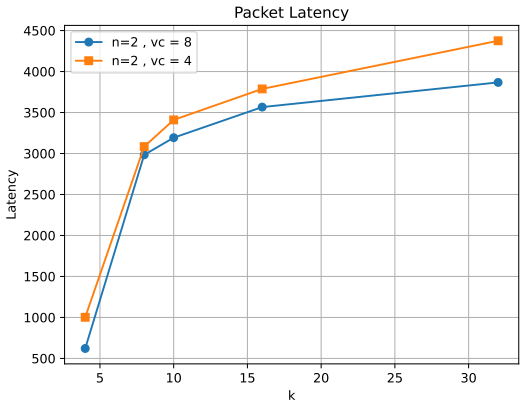
```
    dor_next_torus(r->GetID(), f->dest, 2 * gN,
&out_port, &f->ph, false);
} else {
dor_next_torus(cur, dest, in_channel,
&out_port, &f->ph, false);
}

if(cur != dest) {

  if (f->ph == 0) {
        outputs->AddRange(out_port, vcBegin, vcBegin, 0);
} else {
        outputs->AddRange(out_port, vcBegin + 1, vcBegin + 1, 0);
}
```

در انتها براساس DOR هم مسیریابی انجام شده است و بر روی دو کانال اول مسیریابی مربوط به آن قرار گرفته است. توجه کنید که منطق قسمت آخر دقیقا مشابه الگوریتمی که در کد خود Booksim برای min_adapt_torus استفاده شده است نوشته شده است و برای Deadlock Free کردن است.

فصل ۴

ارزیابی

با اجرای برنامه به ازای مقادیر مختلف k و همچنین دو مقدار متفاوت برای vc نتایج مختلفی را بدست آورده‌ایم که در شکل ۴ـ۱ نمایش داده شده است.



شکل ۴ـ۱: نمودار تاخیر بسته براساس k

همچنین در ادامه یک نمونه از خروجی برنامه به ازای $k = ۱۶, n = ۲, vc = ۴$ آورده شده است.

```
====== Overall Traffic Statistics ======
====== Traffic class 0 ======
Packet latency average = 3118.01 (1 samples)
        minimum = 1022 (1 samples)
```

```
        maximum = 6438 (1 samples)
Network latency average = 382.605 (1 samples)
        minimum = 25 (1 samples)
        maximum = 3561 (1 samples)
Flit latency average = 1110.03 (1 samples)
        minimum = 25 (1 samples)
        maximum = 6593 (1 samples)
Fragmentation average = 0 (1 samples)
        minimum = 0 (1 samples)
        maximum = 0 (1 samples)
Injected packet rate average = 0.0770326 (1 samples)
        minimum = 0 (1 samples)
        maximum = 0.395333 (1 samples)
Accepted packet rate average = 0.0767409 (1 samples)
        minimum = 0.062 (1 samples)
        maximum = 0.0936667 (1 samples)
Injected flit rate average = 0.0770326 (1 samples)
        minimum = 0 (1 samples)
        maximum = 0.395333 (1 samples)
Accepted flit rate average = 0.0767409 (1 samples)
        minimum = 0.062 (1 samples)
        maximum = 0.0936667 (1 samples)
Injected packet size average = 1 (1 samples)
Accepted packet size average = 1 (1 samples)
Hops average = 8.99244 (1 samples)
Total run time 4.39091
```

خروجی زیر هم نتیجه اجرای مدل dimension order به شکل غیر fully adaptive است و مشاهده می‌شود نتیجه کد ما بهتر از مدل پایه است.

```
====== Overall Traffic Statistics ======
====== Traffic class 0 ======
Packet latency average = 2094.91 (1 samples)
minimum = 301 (1 samples)
maximum = 6217 (1 samples)
Network latency average = 272.347 (1 samples)
minimum = 29 (1 samples)
maximum = 3315 (1 samples)
Flit latency average = 527.82 (1 samples)
minimum = 29 (1 samples)
maximum = 6286 (1 samples)
Fragmentation average = 0 (1 samples)
minimum = 0 (1 samples)
maximum = 0 (1 samples)
Injected packet rate average = 0.131471 (1 samples)
minimum = 0 (1 samples)
maximum = 0.557667 (1 samples)
Accepted packet rate average = 0.131405 (1 samples)
minimum = 0.113333 (1 samples)
maximum = 0.150667 (1 samples)
Injected flit rate average = 0.131471 (1 samples)
minimum = 0 (1 samples)
```

```
maximum = 0.557667 (1 samples)
Accepted flit rate average = 0.131405 (1 samples)
minimum = 0.113333 (1 samples)
maximum = 0.150667 (1 samples)
Injected packet size average = 1 (1 samples)
Accepted packet size average = 1 (1 samples)
Hops average = 9.01637 (1 samples)
Total run time 3.65743
```

فصل ۵

نتیجه‌گیری

در این پروژه به پیاده‌سازی یک الگوریتم Fully Adaptive در کنار استفاده از روش Duato پرداختیم. با الهام گرفتن از توابع مسیریابی ابزار Booksim که از پیش روش Duato را استفاده کرده بودند، الگوریتم Fully Adaptive ای توصیه دادیم که یکی از بعدهایی که اختلاف دارد را به صورت تصادفی انتخاب می‌کند و در صورتی که دچار ددلاک نشود از این طریق ادامه می‌دهد و در صورت برخورد به ددلاک، در کانال‌های مشخص شده از روش Dimension Order Routing به کار خود ادامه می‌دهد.

مراجع

[1] W. J. Dally and B. P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[2] N. Jiang, G. Michelogiannakis, D. Becker, B. Towles, and W. J. Dally. Booksim 2.0 user's guide. *Standford University*, page q1, 2010.