Cache complexity

Nikita Frolov
frolov@chalmers.se

Matrix transposition

```
void transpose(int **a, int n) {
  for (int i = 0; i < n-1; i++) {
    for (int j = i+1; j < n; j++) {
      swap(a[i][j], a[j][i]);
    }
  }
}
```

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	2	3	4
17	18	19	20

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	3	4
2	18	19	20
33	34	35	36

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	4	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	4
2	18	19	20
3	34	35	36
49	50	51	52

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	19	20
3	34	35	36
4	50	51	52
5	6	7	8
65	66	67	68

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	19	20
3	34	35	36
4	50	51	52
65	6	7	8
5	66	67	68
81	82	83	84

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	19	20
3	34	35	36
4	50	51	52
65	81	7	8
5	66	67	68
6	82	83	84
97	98	99	100

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	97	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
7	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

113	114	115	116
2	18	19	20
3	34	35	36
4	50	51	52
65	81	97	8
5	66	67	68
6	82	83	84
7	98	99	100

An invisible resource

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	97	113	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
7	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

8	114	115	116
9	10	11	12
129	130	131	132
4	50	51	52
65	81	97	113
5	66	67	68
6	82	83	84
7	98	99	100

Matrix transposition, tiled

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s);
        transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n)
      }
    }
  }
}
```

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	2	3	4
17	18	19	20

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	3	4
2	18	19	20
33	34	35	36

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	4	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	4
2	18	19	20
3	34	35	36
49	50	51	52

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	5	6	7	8	9	10	11	12	13	14	15	16
2	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
3	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	19	20
3	34	35	36
4	50	51	52

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	5	6	7	8	9	10	11	12	13	14	15	16
2	18	34	20	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	34	20
3	19	35	36
4	50	51	52

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	5	6	7	8	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	36	37	38	39	40	41	42	43	44	45	46	47	48
4	20	51	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	34	50
3	19	35	36
4	20	51	52

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	5	6	7	8	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	34	50
3	19	35	51
4	20	36	52
5	6	7	8
65	66	67	68

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	6	7	8	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	34	50
3	19	35	51
4	20	36	52
65	6	7	8
5	66	67	68
81	82	83	84

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	7	8	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

1	17	33	49
2	18	34	50
3	19	35	51
4	20	36	52
65	81	7	8
5	66	67	68
6	82	83	84
97	98	99	100

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	97	8	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
7	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

113	114	115	116
2	18	34	50
3	19	35	51
4	20	36	52
65	81	97	8
5	66	67	68
6	82	83	84
7	98	99	100

An invisible resource (cont.)

Assume a cache that has 8 blocks, 4 words in a block.

1	17	33	49	65	81	97	113	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
7	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

113	114	115	116
21	22	23	24
3	19	35	51
4	20	36	52
65	81	97	8
5	66	67	68
6	82	83	84
7	98	99	100

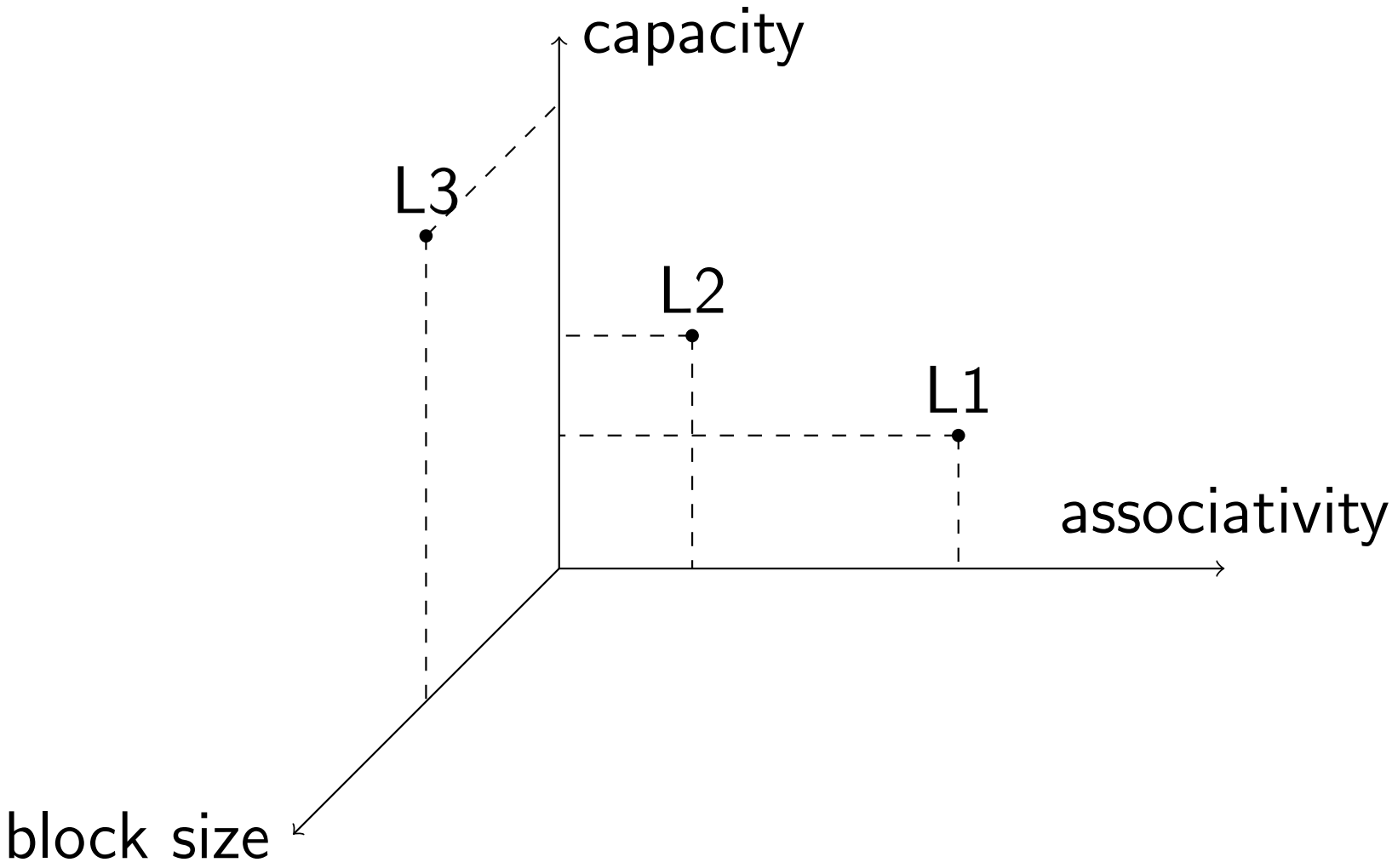
Invisible among invisible

Assume a cache that has 8 blocks, 4 words in a block and is direct-mapped!

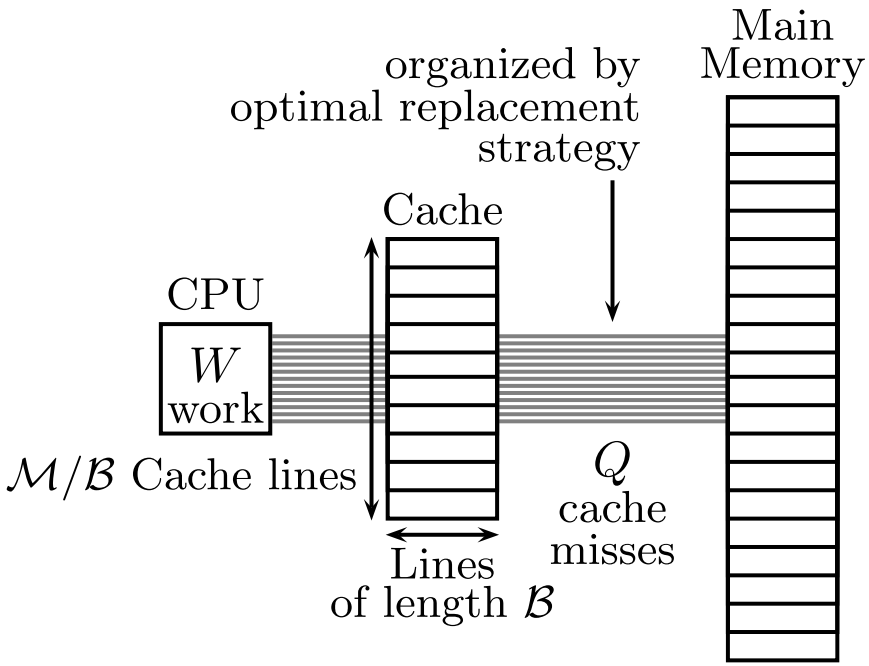
1	17	33	49	65	81	97	113	9	10	11	12	13	14	15	16
2	18	34	50	21	22	23	24	25	26	27	28	29	30	31	32
3	19	35	51	37	38	39	40	41	42	43	44	45	46	47	48
4	20	36	52	53	54	55	56	57	58	59	60	61	62	63	64
5	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
6	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
7	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112
8	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128
129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160
161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176
177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192
193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208
209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224
225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256

113	114	115	116
21	22	23	24
3	19	35	51
4	20	36	52
65	81	97	8
5	66	67	68
6	82	83	84
7	98	99	100

Cache "cube"



Ideal-cache model



- tall ($\mathcal{M} = \Omega(\mathcal{B}^2)$);
- two levels;
- fully associative;
- optimal replacement.

(courtesy of Frigo, M. et al., "Cache-oblivious algorithms", 2012.)

Counting misses

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s); transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n) } } } }
```

- s is chosen to fit two submatrices into cache;

Counting misses

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s); transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n) } } } }
```

- s is chosen to fit two submatrices into cache;
- a submatrix occupies $\Theta(s + s^2/\mathcal{B})$ cache blocks;

Counting misses

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s); transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n) } } } }
```

- s is chosen to fit two submatrices into cache;
- a submatrix occupies $\Theta(s + s^2/\mathcal{B})$ cache blocks;
- $s = \Theta(\sqrt{\mathcal{M}})$ (from tall cache assumption);

Counting misses

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s); transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n) } } } }
```

- s is chosen to fit two submatrices into cache;
- a submatrix occupies $\Theta(s + s^2/\mathcal{B})$ cache blocks;
- $s = \Theta(\sqrt{\mathcal{M}})$ (from tall cache assumption);
- each `transpose` has $Q_t(n) = \Theta(\mathcal{M}/\mathcal{B}) = \Theta(s^2/\mathcal{B})$;

Counting misses

```
void transpose_tiled(int **a, int n, int s) {
  for (int i = 0; i < n/s; i++) {
    for (int j = 0; j < n/s; j++) {
      if (i == j) {
        transpose(a+i*s+j*s), s);
      } else {
        transpose(a+i*s+j*s), s); transpose(a+j*s+i*s), s);
        swap_mat(a+i*s+j*s, a+j*s+i*s, n) } } } }
```

- s is chosen to fit two submatrices into cache;
- a submatrix occupies $\Theta(s + s^2/\mathcal{B})$ cache blocks;
- $s = \Theta(\sqrt{\mathcal{M}})$ (from tall cache assumption);
- each `transpose` has $Q_t(n) = \Theta(\mathcal{M}/\mathcal{B}) = \Theta(s^2/\mathcal{B})$;
- $Q_{tt}(n) = \Theta(n^2/s^2 \cdot s^2/\mathcal{B}) = \Theta(n^2/\mathcal{B})$.

Simulating optimal replacement

For a $(\mathcal{M}, \mathcal{B})$ LRU cache,

$$Q(n; \mathcal{M}, \mathcal{B}) \leq 2Q^*(n; \mathcal{M}/2, \mathcal{B})$$

where
Q assumes LRU, Q^* assumes optimal replacement.

Simulating optimal replacement

For a $(\mathcal{M}, \mathcal{B})$ LRU cache,

$$Q(n; \mathcal{M}, \mathcal{B}) \leq 2Q^*(n; \mathcal{M}/2, \mathcal{B})$$

where
Q assumes LRU, Q^* assumes optimal replacement.

Proof by Sleator, D. D. and Tarjan, R. E. in "Amortized efficiency of list update and paging rules", 1985.

Simulating multi-level hierarchies

A $(\mathcal{M}_i, \mathcal{B}_i)$ cache at the i-th level of hierarchy always contains the same blocks as one-level $(\mathcal{M}, \mathcal{B})$ cache, if:

- LRU;
- elements that share a block on level i also share a block on level $i + 1$;
- level $i + 1$ cache is strictly larger than level i cache.

Simulating multi-level hierarchies

A $(\mathcal{M}_i, \mathcal{B}_i)$ cache at the i-th level of hierarchy always contains the same blocks as one-level $(\mathcal{M}, \mathcal{B})$ cache, if:

- LRU;
- elements that share a block on level i also share a block on level $i + 1$;
- level $i + 1$ cache is strictly larger than level i cache.

Proof idea: level $i + 1$ replaces blocks in the same order as level i.

Simulating full associativity

Simulating full associativity

- direct-mapped: one word that has address starting with *tag* goes into the cache block with *tag*;

Simulating full associativity

- direct-mapped: one word that has address starting with *tag* goes into the cache block with *tag*;
- fully associative: any words go into cache blocks with *tag*;

Simulating full associativity

- direct-mapped: one word that has address starting with *tag* goes into the cache block with *tag*;
- fully associative: any words go into cache blocks with *tag*;
- a *tag* is an id for a cache block, number of *tags* is capacity, large caches have more *tags*;

Simulating full associativity

- direct-mapped: one word that has address starting with *tag* goes into the cache block with *tag*;
- fully associative: any words go into cache blocks with *tag*;
- a *tag* is an id for a cache block, number of *tags* is capacity, large caches have more *tags*;
- for some two capacities, a small fully associative cache will have the same number of *tags* as a direct-mapped one!

Does it always work?

- the commonly used two-way mergesort is oblivious but not optimal;

Does it always work?

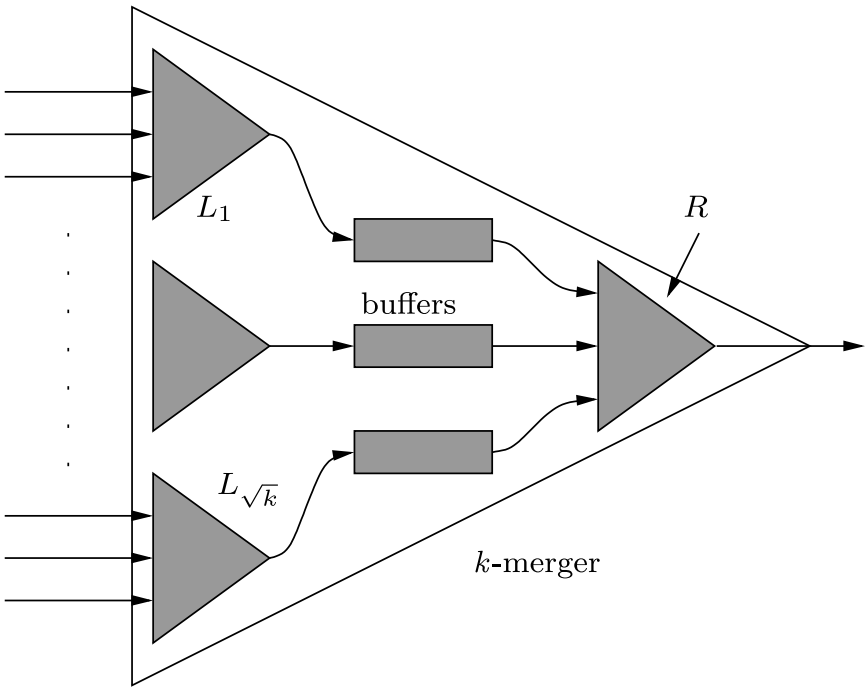
- the commonly used two-way mergesort is oblivious but not optimal;
- the m-way mergesort is optimal but not oblivious.

Does it always work?

- the commonly used two-way mergesort is oblivious but not optimal;
- the m-way mergesort is optimal but not oblivious.

Solution: go from tiling to divide and conquer, as before.

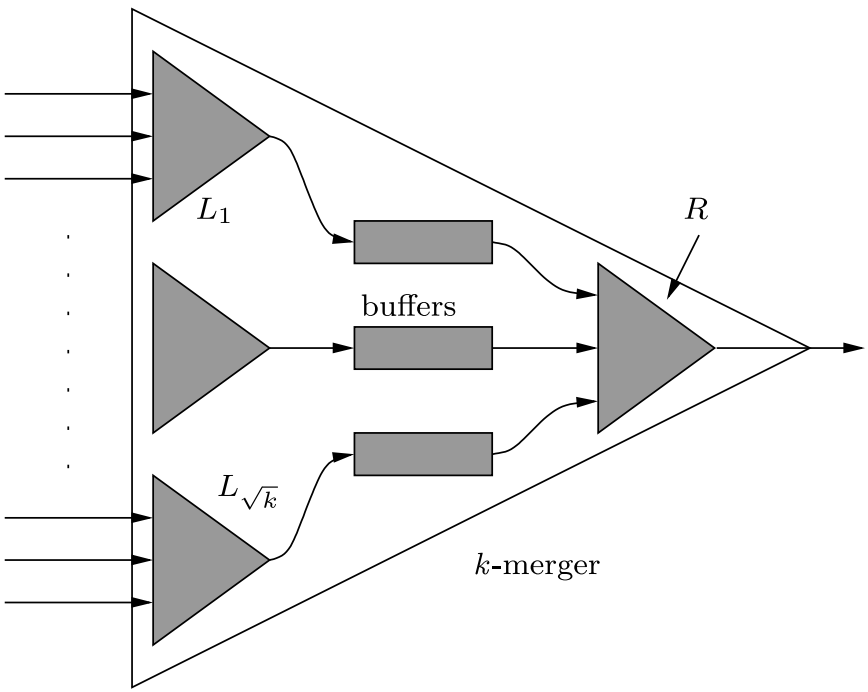
Funnelsort



- input is split into $n^{1/3}$ arrays of size $n^{2/3}$;

(courtesy of Frigo, M. et al., "Cache-oblivious algorithms", 2012.)

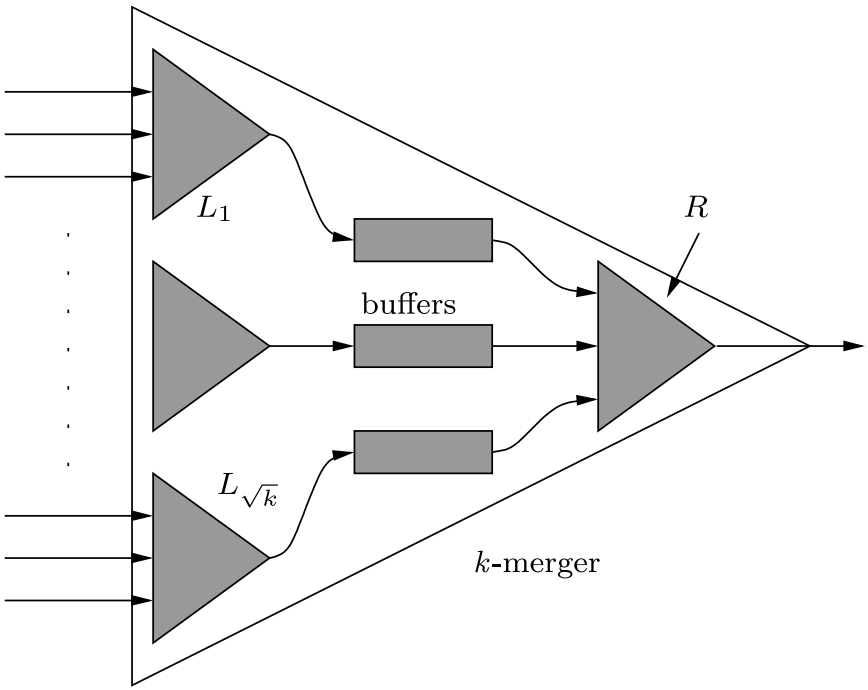
Funnelsort



- input is split into $n^{1/3}$ arrays of size $n^{2/3}$;
- subarrays are recursively sorted;

(courtesy of Frigo, M. et al., "Cache-oblivious algorithms", 2012.)

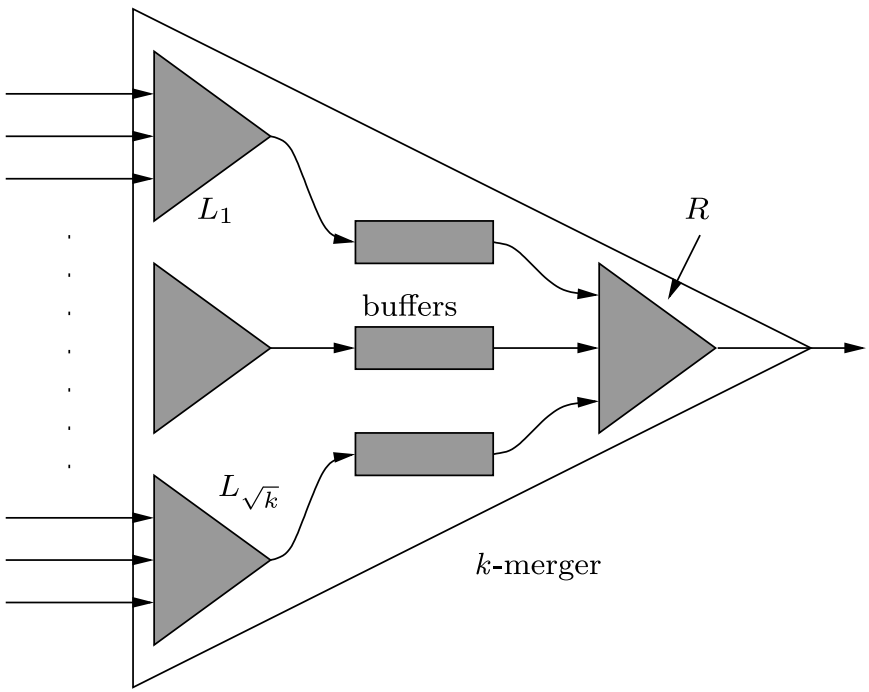
Funnelsort



- input is split into $n^{1/3}$ arrays of size $n^{2/3}$;
- subarrays are recursively sorted;
- $n^{1/3}$ sorted subarrays can be sorted with a $n^{1/3}$-merger;

(courtesy of Frigo, M. et al., "Cache-oblivious algorithms", 2012.)

Funnelsort



- input is split into $n^{1/3}$ arrays of size $n^{2/3}$;
- subarrays are recursively sorted;
- $n^{1/3}$ sorted subarrays can be sorted with a $n^{1/3}$-merger;
- a k-merger is built recursively out of $\sqrt{k}$-mergers.

(courtesy of Frigo, M. et al., "Cache-oblivious algorithms", 2012.)

Funnelsort (cont.)

- a k-merger requires $O(k^2)$ memory;

Funnelsort (cont.)

- a k-merger requires $O(k^2)$ memory;
- $Q_{\texttt{merge}}(k) = O(1 + k + k^3\mathcal{B} + (k^3 \log_{\mathcal{M}} k/\mathcal{B}))$;

Funnelsort (cont.)

- a k-merger requires $O(k^2)$ memory;
- $Q_{\texttt{merge}}(k) = O(1 + k + k^3\mathcal{B} + (k^3 \log_{\mathcal{M}} k/\mathcal{B}))$;
- $Q(n) = n^{1/3}Q(n^{2/3}) + Q_{\texttt{merge}}(n^{1/3}) = n^{1/3}Q(n^{2/3} + O((n \log_{\mathcal{M}} n)/\mathcal{B}) = O(1 + (n/\mathcal{B})(1 + \log_{\mathcal{M}} n)))$.

Conclusion

Remember what's hidden:

- capacity;
- block (line) size;
- associativity;
- replacement policy;
- consistency policy.

Conclusion

Remember what's hidden:

- capacity;
- block (line) size;
- associativity;
- replacement policy;
- consistency policy.

False sharing!

Conclusion

Remember what's hidden:

- capacity;
- block (line) size;
- associativity;
- replacement policy;
- consistency policy.

False sharing!
Divide and conquer is oblivious but not always optimal.

Conclusion

Remember what's hidden:

- capacity;
- block (line) size;
- associativity;
- replacement policy;
- consistency policy.

False sharing!
Divide and conquer is oblivious but not always optimal.
In the real world, recursion cutoff is important and asymptote is not.

Bibliography

- Frigo, M., et al. “Cache-oblivious algorithms.” ACM Transactions on Algorithms (TALG) 8.1 (2012): 4.
- Prokop, H. “Cache-oblivious algorithms.” Diss. Massachusetts Institute of Technology, 1999.
- Cole, R., and Vijaya R.. “Efficient resource oblivious algorithms for multicores with false sharing.” Parallel and Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE, 2012.

Bonus content

Edward Kmett on cache-oblivious data structures in Haskell, Lambda Jam talk and slides